

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE 8 Sept 1993		3. REPORT TYPE AND DATES COVERED Annual: 1 Oct 1992 - 30 Sept 1993	
4. TITLE AND SUBTITLE Combustion Mechanisms of Solid Propellants				5. FUNDING NUMBERS Contract N00014-89-J-1293	
6. AUTHOR(S) E. W. Price and R. K. Sigman					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) School of Aerospace Engineering Georgia Institute of Technology Atlanta, GA 30332-0150				8. PERFORMING ORGANIZATION REPORT NUMBER E-16-689	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 800 North Quincy Street Arlington, VA 22217				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES COR:					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution Unlimited				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report includes results of several ongoing efforts: a) Modifications were made in the computer codes for 2-D diffusion flames to reduce the computation times. A reduction by about 50% was achieved. b) A survey of the literature on combustion of AP-HC sandwiches was made. c) A study of catalysis of burning rate of sandwiches with Fe ₂ O ₃ in the binder lamina was extended to the case where fine AP was added to the binder lamina. The results showed that catalysis of reactions in the region of the burning surface are important (contrary to earlier results with catalysts plus binder alone). d) Studies of melting and decomposition were run on several oxidizers, binders, and combinations using a hot-stage microscope (results are included). e) Preliminary studies were made of mechanisms that cause plateau burning in AP-HC binder propellants. Results indicate that binder melt flow over the AP surfaces is a major factor (confirming past speculation). f) Revisions were made in the laboratory facilities to permit testing at higher pressures and with more hazardous materials.					
14. SUBJECT TERMS Rockets, Combustion, Propellants				15. NUMBER OF PAGES 45	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified		20. LIMITATION OF ABSTRACT UL	

DISCLAIMER NOTICE



**THIS DOCUMENT IS BEST
QUALITY AVAILABLE. THE
COPY FURNISHED TO DTIC
CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO
NOT REPRODUCE LEGIBLY.**

July 9, 1996

Dr. Michael O. Shneier
Chief of Naval Research
Ballston Centre Tower I, Room 607
800 North Quincy Street
Arlington, VA 22217-5660

Georgia Institute of Technology
Atlanta, Georgia 30332-0280
USA
+04•894•3152
+04•853•9378 FAX

Dear Dr. Shneier,

This letter and the attached documents will serve as my final report on project N00014-92-J-1234, "A Case-Based Approach to Creative Design". Our goals were to explore and gain a better understanding of creativity in design so as to be able to create more interesting and creative automated design and problem solving systems and so as to be able to create interactive systems to aid human design and problem solving. Our work has therefore addressed several threads: (1) what constitutes interesting or creative design, (2) designing and testing a variety of software models of processes such as these, and (3) based on these explorations, proposing and implementing tools to aid human designers. What follows will be a summary and bibliography of each area. The papers will flesh that out. If you'd like a longer, more compact report than this, please let me know, and I will work on it during the fall.

1. What constitutes interesting or creative design? We began by observing students in a design class to understand what allowed them to be creative. While they are surely not experts at design, our belief was that by watching them we would learn much about what experts do as well. We believe our observations bear us out. The students, we found, engaged in an iterative cycle that includes three kinds of processes: interpretation or analysis processes, in which the design problem is elaborated and fleshed out and its constraints better defined; generation processes, in which ideas and alternative solutions are generated and refined, and evaluation processes, in which problem descriptions and ideas are critiqued for consistency, completeness, and fulfillment of constraints. Out of this critiquing process, which often includes mental simulation, new constraints are generated and passed on to interpretation processes, and problems are discovered, providing guidance to generation processes on their next cycle. Each type of process might be carried out in several different ways, and we identified several of these. For example, evaluation is sometimes a check of constraints, sometimes carried out by projecting effects using cases, sometimes carried out through mental simulation, and so on. The major differences between novices and experts in addressing problems in a creative way, we believe, are the (1) the tactics available for carrying out each type of process, (2) the available knowledge that the tactics use, and (3) control strategies for deciding which of many possible directions should be followed.

Our case-based approach led us to investigate, in particular, the role that memory and previous experience (whether one's own or that of others) play in interesting design. In our observations, we noticed cases being used by all three types of processes. We also noticed that the process of attempting to recall something from memory sets up cues for recognizing items in the world that might address reasoning needs. For example, the process of trying to remember devices with collapsible tubes as parts led the students we observed to construct in their minds a description of what one of these things might look like if they had ever encountered one. While the devices they were able to recall from their memories were all inappropriate as exemplars for the new design, the description allowed them to easily recognize a toilet-paper holder as one of these when they encountered it in the environment (in this case, a hardware store they were perusing).

Papers:

Kolodner, J.L. and Wills, L.M. (In Press). "Case-Based Creative Design". In Dartnall, T. (Ed.), *Creativity and Computation*, AAAI/MIT Press.

Kolodner, J. L. & Wills, L. M. (1993) Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design. *AAAI-93 Case Based Reasoning Workshop*. July 1993, Washington, D.C. pp. 347-353.

2. What cognitive computational models have we derived from these and other observations? Based on these observations and other observations of experts done outside of this grant, we have developed several cognitive computation models of creativity in design.

In IMPROVISOR, we focused on deriving a computational model that would explain the behavior we saw in the students we observed and that would, at the same time, be consistent with other extant models of memory and problem solving. The model has long-term and working memories that reasoning processes interact with. Its major reasoning processes are the three listed above: interpretation processes, solution generation processes, and evaluation processes. There are several of each kind. Opportunistic and strategic control modules help the system decide what to do next. Each of these modules and their integration is described in the enclosed papers. Perhaps most interesting about the model is that it has allowed us to propose an integrated and parsimonious set of processes that explain how a designer or problem solver can (1) hold some problematic subgoals in abeyance until they can be fruitfully addressed, (2) notice when each of the subgoals has potential to be fruitfully addressed, and (3) notice unexpected opportunities for moving forward both in the results of its reasoning and in the environment. This is accomplished through a combination of a working memory processes that keep track of both active and suspended subgoals along with conditions under which suspended subgoals might become active and perceptual processes that notice when those conditions are met. The processes for noticing opportunities in the environment are interesting in they that build on and use the representations produced by the very processes that allow problem solving and design to proceed, i.e., those that propose what to look for in long-term memory to solve a problem and those that keep track of active, pending, and suspended subgoals. We do not have to propose a new set of processes for noticing the environment; rather, in our model, perceptual processes are informed by memory and reasoning processes already at work. With a model such as this, scheduling goals and tasks associated with unexpected opportunities, whether they are opportunities generated by remembering something, by having some outcome arise from reasoning, or from noticing something in the world, is done by the same scheduler that schedules planned goals and tasks.

The second computer program we have been developing to explore issues in creative design is IDEAL. While IMPROVISOR focused on the architecture that would allow multiple processes to communicate with each other to effect creativity in design, IDEAL focuses in particular on analogical and case-based reasoning processes, with an emphasis on creativity accomplished through non-local modifications to previous designs, cross-domain transfer of design knowledge, and reformulation of problem specifications. Non-local modifications are changes in the arrangement of elements in a design. They are difficult and require ingenuity because in some domains the design elements interact strongly, making changes in their arrangement seem undoable. In cross-domain transfer, experience gained in solving design problems in one domain (e.g., electrical circuits) is

used to solve problems in another (e.g., heat exchangers). Necessary to such cross-domain reasoning is knowledge representations that point out the functional components of designs in the old domain, allowing for recognition of how the components in one domain might match to components in the other. The combination of these processes has been referred to as "constructive analogy." IDEAL proposes such constructs. Problem reformulation means revisions to problem specifications. IDEAL solves these problems through interacting representations and processes. First, it defines the kinds of high-level abstractions, or design patterns, that are useful in facilitating these kinds of reasoning. In particular, it proposes two such patterns as useful: generic physical processes, which capture relationships between design elements, and generic teleological processes, which capture functional relationships. Second, it defines ways in which such abstractions can be learned as part of the cycle of solving a design problem.

These two pieces of work are being followed up in a third computational model called ALEC. In this part of the project, we are looking at the uses of constructive analogy and at the interactions between memory stores and processes and reasoning processes over the course of a long-term design experience. We take as our model Alexander Graham Bell and his invention of the multiple telegraph. As he was inventing this device, he was also engaged in work on several other devices with both acoustic and electrical capabilities. His reasoning moved back and forth between projects, and he was able to use what he learned in working on one project to address the issues in the others. This is a brand new project deriving from our ONR grant and is not specifically funded by it. We include preliminary papers on it because we have come to believe that in order to really understand creativity in design we must look at design as it is happening over a long period of time, and that that requires an understanding of the interactions between the designer and his/her/its world.

While neither of these computer programs is a fully-automated creative designer, each suggests processes, control mechanisms, and memory stores needed in such an automated system. We have published and are about to publish our work in several venues read by those who engage in the design of such systems.

Papers:

Kolodner, J.L. (1994). Understanding Creativity: A Case-Based Approach. In S. Wess, K.D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning, selected papers from the First European Workshop on Case-Based Reasoning*. Kaiserslautern, Germany. Springer-Verlag, 1994. Lecture Notes in Artificial Intelligence, pp. 3-20.

Wills, L. M., Kolodner, J. L. (1994). Explaining Serendipitous Recognition in Design. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 940-945.

Wills, L.M. & Kolodner, J.L. (1994). Towards More Creative Case-Based Design Systems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington, pp. 50-55, August.

Ram, A., Wills, L., Domeshek, E.A., Nersessian, N. & Kolodner, J.L. (1995). Understanding the Creative Mind. *Artificial Intelligence*, Volume 79, Number 1, pp. 111-128, November 1995.

Kolodner, J.L. & Wills, L.M. (submitted). Powers of Observation in Creative Design. Submitted to *Design Studies*, Special Issue on Design Cognition and Computation, Rivka Oxman (Ed.).

Griffith, T.W., Nersessian, N.J., Goel, A. (1996) The Role of Generic Modeling in Conceptual Change. Submitted to the 18th Annual Conference of the Cognitive Science Society, San Diego, CA, 1996.

Nersessian, N.J., Griffith, T.W., Goel, A., (1995). Constructive Modeling in Scientific Discovery. Submitted to AI Journal special issue on Scientific Discovery, Dec. 1995.

Simina, M.D. & Kolodner, J.L. (1995). Opportunistic Reasoning: A Design Perspective. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates (pp. 78--83).

Simina, M. & Kolodner, J.L. (1996). Cases, Reasoning and Bell's Telephone. Submitted to the Eighteenth Annual Conference of Cognitive Science Society, La Jolla, CA, July, 1996.

Bhatta, S.R., (1995). Model-Based Analogy in Innovative Device Design. Georgia Institute of Technology Technical Report No. GIT-CC-96/14.

Goel, A.K., Bhatta, S.R., Stroulia, E. (1996). KRITIK: An Early Case-Based Design System, to appear in *Issues in Case-Based Design*, Maher and Pu (Eds.), MIT Press, 1996.

Peterson, J., Mahesh, K., and Goel, A., (1994) *International Journal of Human-Computer Studies*, Vol. 41, pp. 881-913.

3. Based on these studies, what kinds of tools should we be building to aid human designers? We answer this question based on observations of expert designers (done outside of the context of this grant), observations of student designers, and what we have learned about design processes from the model development and software development we have been doing. First and foremost, creative design requires insight into the ways in which some targeted function has been carried out previously, both successes and failures. This suggests making available to designers libraries of design cases. Each design case should include, as much as possible, the special issues it addressed, how it solved each of those problems, why it did it that way, what else was considered, and what resulted. But we notice that creative use of old designs is different than rote use of them. So it is necessary as well to teach users how to use old designs creatively and to provide "scaffolding," or help, in the software to push them towards using old designs well. Older work suggests that a computer system that can carry out adaptations would be useful. Work on this grant adds to that. IDEAL suggests that we need to help designers develop strategies for rearranging design elements in novel ways, strategies for extracting useful kinds of abstractions from their design experiences, and strategies for mapping design elements in one domain to those in another. IDEAL, IMPROVISOR, and ALEC suggest that case libraries we make available to designers should be broader than just cases in the domain designers are working in. IMPROVISOR and ALEC go farther in suggesting that we should help human designers articulate what their subgoals are, what is holding them up with achieving each, and the circumstances in which each might be revisited profitably.

We are carrying this out in a series of case libraries that we have built and a set of case library authoring tools. DesignMuse was created as an authoring tool before we received the ONR grant. It has been used to implement several case libraries, originally for expert use, in architectural design and airplane design. We discovered, based on research on

creative design, that DesignMuse was insufficient as both a case library and an authoring tool in some ways, especially for design novices. We have therefore designed a new case library tool for design novices. To date, it goes beyond DesignMuse in helping designers understand the relationships between the functions they are trying to achieve in their designs and the ways in which they can be carried out. It still needs to be augmented in the other areas mentioned above. We are trying it in middle school classrooms, and under a current contract, continuing with its development. Canah-Chab concentrates on helping novice designers keep track of the design strategies and problem solving methods they have used and that are potentially available. It is in preliminary form, and its implications are being integrated into the new case library tools we are developing.

Second, our research suggests that designers would profit from aid in keeping track of all of the alternative problem formulations and solutions derived along the way and the different possibilities for moving forward. To that end, we have created a software environment called McBAGEL (soon to be changed to PABLO due to licensing issues), which we have tried in several middle school classrooms, our first attempt at creating such an environment. McBAGEL helps designers keep track of the relevant facts of a situation, ideas for moving forward in solving the design problem, what they still need to learn to move forward, and an action plan. Under a current contract, we are working on augmentations of this piece of software that help them to articulate the pros and cons of different ideas they come up with, play them off against each other, and gain feedback on trying things in several different ways.

Papers:

Narayanan, N.H., Hmelo, C.E., Petrushin, V., Newstetter, W.C., Guzdial, M., Kolodner, J.L. (1995). Computational Support for Collaborative Learning through Generative Problem Solving, *CSCL Proceedings, 95*, ACM Press, pp. 247-254.

Hmelo, C.E., Narayanan, N.H., Newstetter, W. & Kolodner, J.L. (1995). A Multiple-Case-Based Approach To Generative Environments For Learning. Presented at the 2nd Annual Symposium on Cognition and Education, Varanasi, India, December, 1995.

Narayanan, N. H. & Kolodner, J.L. (1995). Case Libraries in Support of Design Education: The DesignMuse Experience. In *Proceedings FIE'95 (Frontiers in Education)*, American Society for Engineering Education (ASEE), Atlanta, GA, November, 1995, pp. 2b2.1.2, IEEE Press.

Goel, Ashok, K., de Silva Garza, A.G., Grue, N, Recker, M.M., Govindaraj, T, (1995). Beyond Domain Knowledge: Towards a Computing Environment for the Learning of Design Strategies and Skills, Cognitive Science Technology Technical Report (Georgia Institute of Technology) No. GIT-COGSCI-95/02.

In addition to work on design creativity over the long term, my newer work concentrates in the areas of education and educational technology, especially helping students learn from design experience. Ashok Goel's newer work concentrates on constructive analogy and scientific discovery. We continue to build on what we've learned from work on our ONR

grant in all of these projects, and I think you would be pleased to see how our work on creative design is influencing and contributing to those investigations. We receive inquiries from around the world on our creativity work; there is much interest in it. We will be happy to have you visit us to see the creative software we've designed and to discuss our latest work.

Sincerely,

A handwritten signature in black ink, reading "Janet L. Kolodner". The signature is fluid and cursive, with a long horizontal flourish extending to the right.

Janet L. Kolodner,
Professor Computing and Cognitive Science

cc: Ashok Goel

CASE-BASED CREATIVE DESIGN

Janet L. Kolodner and Linda M. Wills

College of Computing

Georgia Institute of Technology

Atlanta, Georgia 30332-0280

jlk@cc.gatech.edu, linda@cc.gatech.edu

Abstract. Designers across a variety of domains engage in many of the same creative activities. Since much creativity stems from using old solutions in novel ways, we believe that case-based reasoning can be used to explain many creative design processes.

1. Introduction

Designers across different domains perform many of the same creative activities, whether they are involved in designing artifacts or processes. These activities can be described by contrasting them to routine design activities. In general, routine design repeats old designs in obvious ways, adapting them by well-known and often-applied adaptation strategies. Routine design assumes a completely specified problem is given and little effort is applied to elaborating or designing a feasible specification.

The kind of design we call creative, on the other hand, includes a process of "designing the design specification" (Tong, 1988), going from an incomplete, contradictory, and underconstrained description of what needs to be designed to one with more detail, more concrete specifications, and more clearly defined constraints. Creative design also often includes a process of generating and considering several alternatives, weighing their advantages and disadvantages, and sometimes incorporating pieces of one into another. It involves using well-known design pieces in unusual ways or modifying well-known designs in unusual ways. Creative designers frequently engage in cross-domain transfer of abstract design ideas. They also often recognize alternative uses or functions for common design pieces (e.g., using a styrofoam cup as a boat).

Figure 1 gives a rough sketch of the main processes we hypothesize to be involved in creative design and how they interact with one another. The designer continually updates the design specification as well as a pool of design ideas under consideration. Each alternative generated is evaluated to identify its advantages and disadvantages and to check that it satisfies the constraints in the current design specification. A key part of evaluation is "trying out" the alternative (e.g., through experimentation or mental simulation). This generates a more detailed description of the alternative, including the consequences of its operation and how environmental factors affect it.

Evaluation drives both the updating of the design

specification and the modification and merging of design alternatives. It raises questions of legality or desirability of features¹ of a design alternative and it detects contradictions and ambiguities in the specification. The resolution of these questions, contradictions, and ambiguities serves to refine, augment, and reformulate the design specification. On the generative side, evaluation identifies advantages and disadvantages of alternatives which often suggest interesting adaptations or ways of merging alternatives. Also, sometimes the description of a problem noticed during evaluation can be easily transformed to a description of how its solution would look.

The three processes interact opportunistically. The generative phase, guided by critiques from the evaluation phase, watches for opportunities to merge or adapt design ideas to create new alternatives. The design specification is incrementally updated as ideas are tested and flaws or desirable features become apparent.

The continual elaboration and reformulation of the problem (i.e., the design specification) derives abstract connections between the current problem and similar problems in other domains, facilitating cross-contextual transfer of design ideas. Continual redescription of what the solution (i.e., the evolving design) looks like primes the designer for opportunistic recognition of alternative functions of objects.

This paper describes the nature of these processes and proposes ways of modeling them. Since all three processes rely heavily on previous design experiences, case-based reasoning (Kolodner, 1993) can play a large role in modeling them. Research in case-based reasoning has provided extensive knowledge of how to reuse solutions to old problems in new situations, how to build and search case libraries (for exploration of design alternatives), and how to merge and adapt cases. Many of the activities of creative designers can be modeled by extending routine problem solving processes that exist in current case-based systems.

¹The features of a design alternative are not only its structural characteristics and physical properties, but also relations between combinations of features.

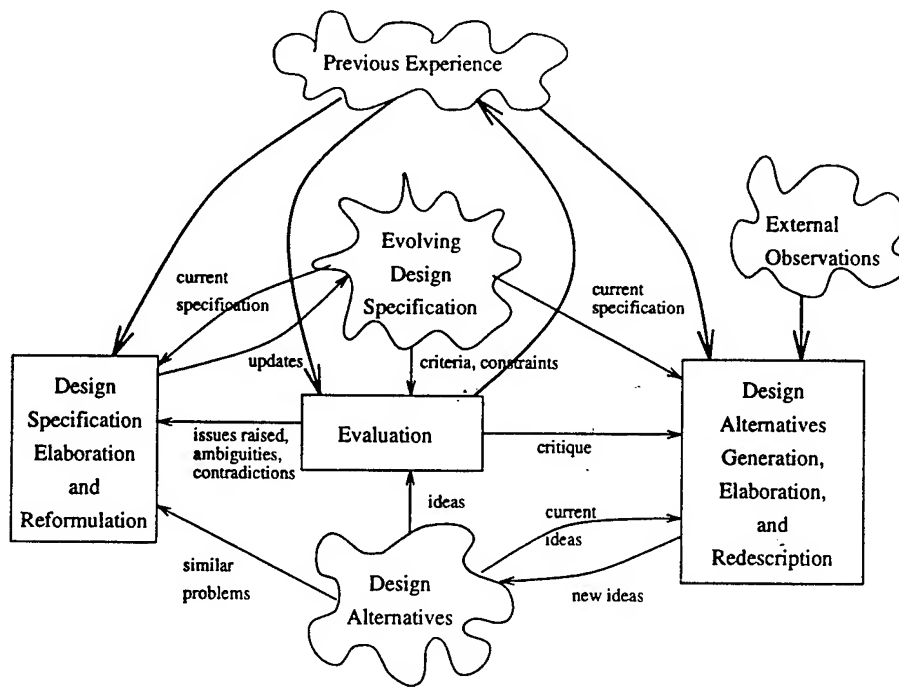


Figure 1: Rough sketch of creative design processes.

We give examples to illustrate these activities, which we have collected in studying the problem solving reports and protocols of designers in a variety of design disciplines. These include software design, meal planning, science lesson planning, architectural design, and mechanical design. Many of the anecdotes related in this paper come from an exploratory study we conducted of a student mechanical engineering (ME) design project. The design task was to build a device to quickly and safely transport as many eggs as possible from one location to another. The device could be constructed from any material, but had to satisfy a set of size, weight and cost restrictions. The initial description of the problem was vague, ambiguous, and incomplete, requiring a great deal of elaboration and reformulation. One of us participated in the seven-week project as a member of a four-person team. Active participation in the project allowed us to become immersed in the issues the students were dealing with and to openly converse with the students at all stages of the design as a useful team member, rather than as an outside observer. This led to many of the insights described in this paper.

2. Specification Refinement

Design specifications are rarely well-defined. In general, they are incomplete, leave many different ways to solve a problem, and are often unnecessarily overconstrained. An important part of design is redefining the design specification. This includes elaborating the constraints and criteria the design should satisfy and extensively restructuring the problem (Goel and Pirolli, 1989).

2.1 ELABORATION

In general, a designer has goals and guidelines that are not in the initial design specification itself but whose violation or achievement can be noticed. For example, a meal planner might like meals to be easy to prepare, but may not include this in every design specification. Goel and Pirolli (1989) identify several classes of constraints that are of this nature, including domain-specific technical constraints (such as structural soundness), legislative constraints (such as building codes), common sense, pragmatic constraints (for example, "short construction time" or personal safety), and self-imposed, personal preferences (such as "not spicy").

Elaboration involves making these constraints and criteria explicit, consistent, complete, and unambiguous. We hypothesize that this is driven in part by the process of evaluating each alternative generated so far. Evaluation drives elaboration by satisfaction or dissatisfaction with an alternative and by an inability to evaluate. Elaboration is also driven by an inability to generate satisficing alternatives, and by opportunity. These are discussed in this section.

Many design alternatives arise from remembering or looking for solutions to old design problems. Such design cases are, in general, similar to the new situation on important dimensions, but are more complete. Additional aspects fuel elaboration by bringing up new constraints or criteria to consider. They are evaluated for applicability to the current design problem. The results are used to update the design specification: if the case is applicable, more detailed constraints are added; if the case is rejected, constraints are added to prohibit the aspects that were unacceptable.

For example, while designing a manufacturing research center on the Georgia Tech campus, Terry Sargent visited existing manufacturing centers and precision engineering laboratories around the world. Examining these options helped him decide what criteria and constraints were important and how to prioritize them. One technical center he visited has flexible utilities which can be tapped into at any location in the building (e.g., an air duct can be added anywhere). On the other hand, all of its research laboratories are internal and the building is too dark. From his examination, Sargent formed a wishlist of constraints for his building to satisfy, including having flexible utilities, external offices, and letting in plenty of sun.

This illustrates two of the ways evaluation drives elaboration: by satisfaction and by dissatisfaction with an alternative. A third way is by an inability to evaluate. This occurs when there is a lack of information in the specification to confirm or reject the legality and desirability of features. It suggests new constraints and criteria to add or existing constraints to disambiguate.

An example arose in the ME design project, where a possible starting location of the device was from the center of a wading pool of water. The team discussed the idea of launching an egg-carrying device from a model battleship. To determine whether this was legal, they needed to know whether it was all right to leave parts of the device behind as it operated. The answer to this question added to the problem description.

Elaboration is also driven by an inability to generate satisficing alternatives. In general, this results in relaxing constraints (i.e., making a compromise). In the ME project, the students originally wanted to carry more than a dozen eggs, but could think of no design ideas that would allow a large number of eggs to be carried safely, given the amount of protective cushioning required and the space restrictions. This led the students to relax their preference for the device to have a high egg-carrying capacity.

Finally, elaboration is driven by opportunity. If the evaluation process is aware of the designer's other goals, it can be opportunistic. For example, a meal planner whose immediate goals were to use leftover rice for dinner remembered a breakfast dish. Since she needed to eat breakfast too, she decided to relax the dinner goal and use the rice for breakfast. This required reasoning about priorities and alternative ways of doing things. If rice is the only thing of substance available for dinner, then using the rice for tomorrow's breakfast is a poor idea. If, on the other hand, there are plenty of other things available for dinner and/or the eater didn't really want to eat rice anyway, then using it for breakfast solves two problems. So, evaluation may allow a reasoner to opportunistically realize that a solution is good, even though it does not fit the design specification. This can lead to a change in the relative importance of goals and constraints in the current problem description.

2.2 REFORMULATION

Another major activity in designing the design specification is reformulating the problem – redescribing the problem so that the solution is easier to find. There are several ways alternative views of a problem can be generated.

One way stems from making a design alternative more concrete, e.g., by mentally visualizing it or acting it out. The more detailed description of the solution sometimes suggests a new description of the problem. For example, in the ME design project, while considering how to move eggs out of a pool of water, one student made an analogy to submarines launching missiles. He acted out the launch with his pen as he spoke. His description reminded another student that submarines launch missiles one at a time. This led to reformulating the problem from one of moving all eggs as a group to moving eggs individually.

It is an open question exactly how a more detailed description of a solution can suggest a reformulation. It may be that the visualization of the submarine launching is making assumptions explicit. It is challenging constraints that have been inherited from previously considered options, but which are not essential, e.g., the constraint on how many eggs should move at once.

Another problem reformulation technique is to explore and stretch the problem constraints and exploit any loop-holes found. For example, a designer trying to "design a building between two buildings" (Goel and Piroli, 1989) might ask how close the middle building can be to the two adjacent buildings. By taking closeness to the limit, the designer can reformulate the problem as "connect two buildings together."

Finally, a third way an alternative view of a problem sometimes arises is from realizing part of a solution and then reducing the problem to making that happen. For example, Maier (1931) describes an experiment in which subjects were given the problem of connecting together two strings that hung vertically a large enough distance apart that the person could not hold one string and reach the other. The solutions depend on describing the problem in different ways: "how to make one string longer," "how to make one string stay in middle and bring the other string to it," "how to extend my reach to pull one string to the other," and "how to make one string move to the other." Maier showed that subtly giving the hint of making one string sway often helped the subjects come up with the fourth reformulation (which led to the solution of tying a weight to the string, making it swing like a pendulum toward the other string).

Turner (1991) provides an initial attempt to model the problem reformulation process, which he implemented in a program called MINSTREL. Turner proposes a case-based model of creative reasoning in which a given problem is transformed into a slightly different problem and then used as a probe to a case library. A recalled solution to the new problem is then adapted back to the original

problem (using solution adaptations that are associated with the problem transformations). A set of "creativity heuristics" is used to transform the problem. Examples include generalizing a constraint (and perhaps suspending it altogether), and adapting a constraint to require a related, but slightly different outcome (e.g., injuring instead of killing).

Unfortunately, MINSTREL does not address important focus of control problems. For example, what guides the problem reformulation? Which features or constraints should be varied? Figuring out what to change and how seems to be a major part of recasting problems. We believe that incorporating feedback from the evaluation of proposed alternatives can provide focus.

3. Idea Exploration

Generating design alternatives is an incremental, opportunistic process that is tightly interleaved with specification refinement and evaluation. Three primary ways in which ideas are put on the table for consideration are retrieval of previous design experiences, recognition of current experiences or design pieces in the current environment as potential solutions, and modifying or combining existing options to produce new ones.

3.1 REMINDING

An expert designer knows of many design experiences, accumulated from personally designing artifacts, being given case studies of designs in school, and observing artifacts designed by others. Our observations and analyses lead us to propose that reminding of these experiences is crucial to generating design alternatives. When a design experience is recalled, it suggests a potential solution that can be critiqued with respect to the new problem, adapted to meet the needs of the new situation, or merged with other proposed solutions.

Designers frequently choose an already well-known framework (or generic case) for a problem and then fill it in. Reusing solution structures in this way allows designers to avoid recomputing useful compositions of design pieces. We call this process "framing a solution." The framework provides the glue holding the pieces of the design together. The creativity comes in filling in details and in dealing with inconsistencies when merging alternative pieces.

Such framing occurs in domains, such as bridge design and engine design, where well-known frameworks exist and where constraints holding the pieces of problems together are quite complex. In software engineering, frameworks exist as widely-used computational fragments, called clichés (Rich and Waters, 1990). Johnson and Foote (1988) have defined a similar notion of "frameworks" for reuse of object-oriented software. In other domains, such as architectural design, creating the framework is a primary piece of the creative process. This involves deciding which aspects of a problem specification are most important to deal with first and inferring structural aspects of a solution from them.

We hypothesize that unorthodox design alternatives tend to come from non-obvious reminders. Some are based on abstract similarities, resulting in cross-contextual reminders. Other reminders are based on derived or computed features rather than available ones.

If reminding is so important to generating alternatives and if it requires derived or abstract features, we must determine which kinds of derived features tend to be most useful for design, whether there is a set of derived features that is common to design across domains, and when those features get derived.

In her investigation of story writing, Dehn (1989) stresses the importance of reusing old ideas in new ways. Of particular importance is having processes that are able to generate multiple alternatives for several parts of a problem and put them together in unusual ways. This requires processes that can search memory for things that might be represented in a way that is different from the representation of the current problem. Old cases must be seen in a new light.

Recent studies of creative problem solving protocols (Kolodner and Penberthy, 1990) suggest that anticipatory indexing is not sufficient to fully explain retrieval. Features that were not salient at the time a case was experienced might be important for retrieval in the current situation. Drawing new, abstract connections might be a result of re-indexing cases in terms of what is now relevant or important. We hypothesize that by continually updating the design specification, designers derive abstract connections between the current problem and similar problems (possibly in other domains). These abstractions can be used to see previous cases differently.

Selfridge (1990) claims that people tell stories to re-index them under new generalizations that have been learned since the story was first acquired. A key open question he identified is how does a person know what stories to tell? One possibility is that they are the ones the person is reminded of or has experienced recently. The person may have been reminded of them through a different set of features than the generalized features they are re-indexed under. While working on a design problem, designers often perform sensitized recognition of current design options and objects in their environment and they continually re-examine and re-index all ideas recently brought up or experienced. This is discussed further in the next section.

Retrieval can be automatic or strategic (i.e., based on intentional elaboration strategies that help jog a designer's memory). Strategic retrieval is promoted by design team communication. Team members describe abstract ideas to each other in terms of concrete examples, analogies, and metaphors. Trying to recall an appropriate example often involves applying elaboration strategies to an index. For example, the person might reflect on "where have I seen something like this before?" and "in what situations might I have seen something like this?" This often results in identifying opportunities to

reuse existing objects or devices in the current design.

Team communication plays an additional role in idea generation: ambiguity in communication is generative. In general, when working together, team members try to recognize and understand each others' ideas, plans, and goals from their actions, words, and sketches. Sometimes there is ambiguity in the interpretations which often helps generate more ideas (increases fluidity of concepts) and can lead to function sharing optimizations. Goel (1992) studied the generative role ambiguity plays in informal sketching. In our informal study, we have noticed that interaction among multiple designers amplifies its effects.

3.2 SENSITIZED RECOGNITION

As designers become deeply involved in design problems, they start to recognize objects in their environment as solutions to parts of the design problem. Often the objects are seen as having alternative, unusual functions or uses.

For example, in the ME design project, the students were considering using a spring launching device and went to a Home Depot (a home improvement store) to look into materials. While comparing the strengths of several springs by compressing them, they noticed that the springs bent. One student mentioned that if they were to use springs, they would have to encase the springs in collapsible tubes to prevent bending. Later, as they walked through the bathroom section of the store, they saw a display of toilet paper holders. They immediately recognized them as collapsible tubes that could be used to support the springs.

The key to sensitized recognition is refining the description of the solution. The process of critiquing proposed ideas often yields descriptions of what an improved solution would look like: what properties it would have, what function it should provide, and what criteria it satisfies. This primes the designer to opportunistically recognize possible solutions in observations of the external world and in recently considered design options.

3.3 ADAPTATION

Previous work has looked at adapting old solutions to fit new problems. In creative design, it sometimes makes sense, in addition, to adapt one's goals to fit an old solution rather than changing the old solution to fit the new problem (e.g., using rice for breakfast rather than dinner). Previous work (Hinrichs 1992) has looked at routine adaptation strategies (e.g., deletion, addition, substitution) but not at use of "off-the-cuff" ones (i.e., those developed in response to a particular problem). Some of these arise from examining a causal model, some from adapting well-known adaptation strategies, and some from applying well-known adaptation strategies in novel ways. For example, novelty can result from substituting something different than the usual thing or from relaxing well-known structural constraints.

3.4 MERGING

In routine design, parts of several designs are often merged, but in general, the parts are non-overlapping (e.g., dessert from one meal might be used with a main dish from another meal). In more novel design, several suggestions for solving the same part of a problem might be merged to come up with a solution (e.g., in deciding to have salmon fettuccine and salad for dinner, a meal planner might have remembered three previous cases, a meal with fish, a one-dish meal and a pasta meal, and merged desirable features from each).

Merging pieces of several solutions into one design is relatively simple if the pieces are consistent with each other. Either a previous case will suggest a way of combining them, an adaptation heuristic will know how or combination will be obvious. Merging is more complex when the pieces are not obviously consistent. We have two hypotheses about how creative merging of several alternative solutions might work. First, some adaptation heuristics might exist that can provide general guidelines and suggestions for non-routine merging. Second, cases from other domains may provide guidelines and suggestions for non-routine merging. The challenges here are to find the adaptation heuristics and to discover the descriptive vocabulary that allow cross-contextual reminders of the appropriate kinds to take place.

3.5 FUNCTION SHARING

Often function-sharing optimizations arise from merging within the same design. This occurs when an existing part of the design can be seen to fulfill another purpose. (This is a special case of sensitized recognition.) An interesting form of this type of merging occurred in the ME design project. The students had decided to use a cylinder to carry the eggs. One student related an episode from the children's science TV show *Beakman's World* that had caught her eye as she was flipping through channels. The episode showed how to make a coffee can that rolled back to you when you rolled it away. It attached batteries as weights to rubberbands, strung through the center of the can. The weights caused the rubberbands to get wound up as the can rolled. As the rubberbands unwound, they caused the can to roll back to the starting location. The students discussed whether this could be modified for use in their design (e.g., wind the rubberbands up and let their unwinding launch the device). They criticized the rubberband and battery part for taking up too much space and for adding too much weight, since the task had strict space and weight restrictions. One student then suggested the interesting optimization of letting the eggs themselves be used as the weights. This alleviated both the space and the weight problem. One aspect that was non-routine about this is that the student looked beyond the structure of the device to its cargo to find what to share.

4. Evaluation

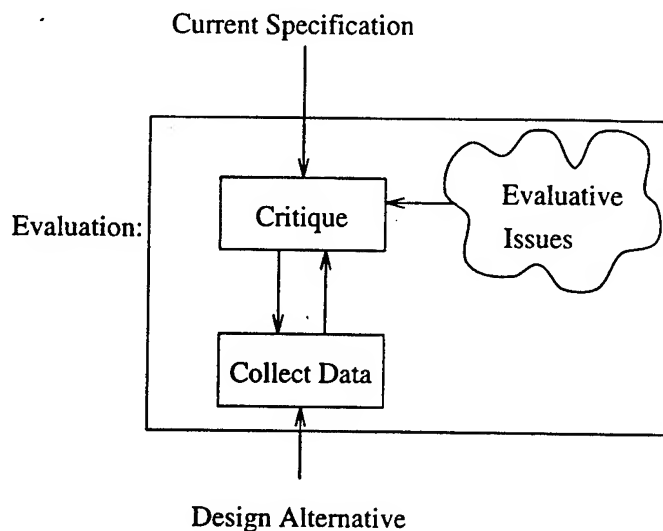


Figure 2: Processes involved in evaluation.

The evaluation process checks each design option that is generated against the current design specification. It forms a critique, identifying how well the option satisfies the constraints or how badly it fails. It also notices questionable features whose desirableness or legality are unknown. In addition, it raises evaluative issues and guidelines that are not found in the current specification, but which are based on the designer's experience. Some of these are always raised. For example, in algorithm design, issues of correctness, completeness, and time and space efficiency are routinely considered. Others (e.g., elegance) are recalled or derived based on features of the alternatives examined.

This information is used by both the specification refinement process (elaborate and reformulate) and the idea exploration process (generate, elaborate, re-describe). The issues raised point out opportunities to augment or refine the design specification. The pros and cons that are described in the critique of a design idea are used by the idea exploration process to compare the idea to other options, merge and adapt alternatives, and improve promising ideas.

We view evaluation as consisting of two interacting processes, as shown in Figure 2. One process critiques the design alternative on the basis of the current specification and the evaluative issues. The features examined in this critique are not only the structural characteristics of the design artifact, but also information about how it behaves, the consequences of its operation, and how environmental factors affect it. The second process collects this information by performing simulations and experimentation.

In the ME design project, the students often mentally simulated proposed options and checked the results. For example, when the idea of launching each egg individually rather than as a group was considered, the students imagined that the eggs would all land on top of

each other which could cause breakage and an unstable target spot. Identifying this problem through mental simulation led to an adaptation of the proposed solution which was to rotate the launch mechanism so that the eggs would each land in a different location.

In addition to simulating the proposed option in the general case, designers also propose hypothetical situations to simulate. For example, the ME students asked, "What if it is raining on the day of the competition?" and "What if the terrain the device must traverse is rough or steep?" Simulations of hypothetical situations test the robustness of the solution. The hypothetical situations pertain to all phases of a design artifact's lifecycle, including its construction and maintenance, as well as its use. For example, a designer might try to imagine someone trying to repair some part of the design that is vulnerable to failure and consider whether the part is accessible for maintenance.

Concrete experimentation of design alternatives is a valuable way of collecting data. Some aspects and outcomes of an option only become apparent through real-world testing. For example, during the ME design project, the students tested the ability of potting sponge (used in floral displays) to cushion eggs. When an egg was placed in it and dropped, the sponge compressed to a powder, decreasing its protective ability and reusability. This led the students to search for a material that did not permanently compress and was reusable.

Some simulations or experiments might be proposed by the critiquing process when it requires additional information about the design option to judge its strengths and weaknesses. Some hypothetical situations used in simulation might be associated with evaluative issues raised in critiquing the design option.

4.1 EVALUATIVE ISSUES

While critiquing a given design option, a designer considers general evaluative issues that the designer's experience recommends looking into, in addition to how well the option fits the current design specification. There are at least three classes of evaluative issues that designers routinely raise (Kolodner and Penberthy, 1990).

One is *function-directed*. For example, the purpose of recipe creation is to create something that can be eaten, so some questions arise from the concept of edibility. These focus on the taste and appeal of a dish to see if it is edible.

Another class is *derivation-driven*: previous solutions provide a rich and important source of issues if the considerations taken into account in creating them are saved. Consider, for example, the task of trying to decide if tofu can be substituted for cheese in tomato tart. One way the right evaluative issues can be derived is by recalling another case where tofu was to be substituted for cheese. Concerns in that case are likely to be concerns in the current one, too. For example, if in the previous case, the texture of tofu was compared with the texture

of the original ingredient, the reasoner might then ask about texture in the current case.

Finally, some questions are *outcome-related*. Previous design cases can be used to project or derive the outcome of the current one. For example, as part of the ME design project, a proposed launch mechanism was considered that consisted of a plastic fish tank base and two toilet paper holders (which provided a spring mechanism). The two holders were attached to the base via plastic prongs protruding from one side of the base. One of the students was concerned that the prongs were vulnerable to breaking, particularly if the springs inside the holders were replaced with stronger springs. She recalled similar plastic prongs had held a protective covering on her stereo speakers, but they had broken off of one speaker when it fell at an angle. The proposed design option was used as a probe to memory to see if instances are already known of it or a similar solution failing. By recalling the stereo speaker case, the students raised the question of whether the proposed design was vulnerable in the same way. It also suggested a hypothetical situation in which to simulate the proposed design: what happens if we provide a large side-ways force to the prongs? Thinking about how this could arise led the students to think about what would happen if stronger springs were required.

Case-based projection can bring up outcome-related issues relevant to any phase of a design's lifecycle, besides its normal use, including its construction and maintenance phases. For instance, one of the buildings Terry Sargent examined when designing the Georgia Tech manufacturing research center was the Pompadour center, which has all of its mechanical systems showing. He wanted to borrow this idea for its symbolism, but in talking with the managers of the center, he found out that this feature made it difficult to maintain the building. This led him to question whether the same maintenance problems will come up in his design.

5. Discussion

Creative designers operate in a rich context of ideas, some recalled from previous experiences, some recognized in the current external environment, and some generated from adapting or putting together recently considered ideas. An important part of this rich context is concreteness. Details fuel evaluation, which is central to elaborating and redescribing both the problem and the solution. These come from reasoning about specific design cases, which include many additional details besides those aspects that originally brought the case to mind. They also come from experimentation, testing, visualizing, and simulating the solutions.

This suggests three important ways to assist creative design. One is by placing the designer in a rich environment containing concrete design artifacts or detailed descriptions and simulations of existing design artifacts. Another is by facilitating evaluative procedures

and proposing hypothetical situations covering the artifact's entire lifecycle. The third is by assisting the designer in reformulating and redescribing what is needed, what constraints or criteria need to be satisfied, and what the solution would look like.

5.1 OPEN CONTROL ISSUES

Our exploratory studies of designers have given us insights into the primary activities involved in creative design. However, many open issues remain. Most center around the underlying control of the various processes and their interactions.

Specification refinement. A key activity in designing the design specification is incrementally bringing evaluation criteria and new problem constraints into focus. This is largely driven by evaluation. An open question is how does noticing a feature of a design option that is either satisfactory, undesirable, or whose status is unknown (due to failure to evaluate) lead to an elaboration of the current specification? One possibility is that it can be guided by the mechanism that detected the questionable feature. For example, one way to detect a problem in a proposed solution is by case-based projection: recalling a failure in a similar solution. This previous case might provide suggestions for fixing the current problem specification. Failure to determine the legality of a feature could point to augmentations to the specification that would push the confirmation or rejection through to completion.

Another important question is: during problem reformulation, how is the designer's attention drawn to particular constraints to explore and stretch? There seems to be give and take between reformulation and evaluation. Evaluation can home in on what is ambiguous or vague in the problem specification and try to take advantage of new views that result from relaxing or pushing the limits of the constraints. Also, when the need to compromise arises, conflicting constraints come into focus and the designer considers how they can be varied. On the other hand, reformulation of the specification can provide additional or improved evaluative measures to strengthen evaluation.

Idea exploration. The critique of proposed solutions guides idea exploration. Of several solutions under consideration, one might be more appropriate than the others or several might each contribute to a solution. Evaluative procedures must be able to evaluate each individual alternative by itself as well as in light of the others. Several open questions arise: How is relative importance among the criteria decided? How are preferences among alternatives made? How does weighing advantages and disadvantages suggest useful adaptations and mergings?

Recalled cases seem to be important here. They suggest solutions, frameworks, design strategies and design philosophies, which can provide constraints with which to evaluate a solution and the preference criteria with

which to prioritize the constraints. This also facilitates reformulating the specification, making trade-offs, and relaxing constraints. There may also be general and domain-specific strategies for setting priorities that we haven't discovered yet.

Evaluation. An important and open question is how does the evaluation process know which aspects of a design alternative to focus on? Of all the data collected during simulation and experimentation, which subset is interesting? For example, which data is likely to suggest updates to the design specification or adaptations that lead to new ideas?

Evaluative issues that designers always raise tend to focus on particular features. At the same time, some features seem to draw attention to particular evaluative issues that might not have been considered otherwise. Some of the features are more distinctive or odd and these seem to index directly into the set of implicit criteria held by the designer. An example arose in the ME design project. While testing how well various types of spongy material cushioned eggs when dropped from two stories, a person walked by who had done a design project which also involved protecting an egg from breaking on impact. He said he wrapped the egg in a sponge soaked in motor oil and then stuffed it in a Pringles can (a narrow cardboard cylinder in which potato chips are stacked). One of the aspects that was new about this case, compared to the ideas the students had been considering is the idea of soaking the sponge in motor oil. Focusing on the motor oil aspect reminded the students of their personal preference that the device be clean. The motor oil aspect seemed to index directly into the cleanliness criterion.

Overall Control. Other open questions pertain to how designers decide when to expend effort in one process versus another. For example, when should quick adaptations of existing solution ideas be tried and when should the designer step back and reformulate the problem. One observation we made in the ME student design project was that when flaws were noticed, the students usually preferred to redescribe the solution rather than elaborate or reformulate the problem specification. The students described what was needed in terms of how the structure of the device should be modified to fix the problem (e.g., "the launch mechanism must rotate" or "the springs should be in a collapsible tube") as opposed to describing what function or behavior is desired (e.g., "the eggs should each land at different target locations" or "provide side-to-side support to springs"). The students usually tried to adapt the offending feature, before reformulating the problem. Only when quick adaptations to the solution were not sufficient did they step back, look at the essential problem constraints these specific structural solutions were solving, and then reformulate the problem or find other solutions that could also satisfy these constraints. This is reasonable, since it is cheaper to make small changes to an evolving design solution

than to completely reformulate the problem. We need to look for additional types of heuristics people use to control their reasoning processes.

Acknowledgments. We would like to thank the members of the mechanical engineering design team, Otto Baskin, Jon Howard, and Malisa Sarntinoranont, for their invaluable cooperation. We thank Farrokh Mistree for allowing us to participate in his mechanical engineering project course and for many thought-provoking discussions throughout. We also appreciate the insightful discussions we have had with Sambasiva Bhatta, Ashok Goel, Colin Potts, Mimi Recker, Spencer Rugaber, and Terry Sargent. This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234.

References

- Dehn, N.J.: 1989, *Computer Story-Writing: The Role of Reconstructive and Dynamic Memory*, Ph.D. Thesis. Yale University.
- Goel, V.: 1992, 'Ill-Structured Representations' for Ill-Structure Problems. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 130-135.
- Goel, V. and Pirolli, P.: 1989, Motivating the Notion of Generic Design within Information Processing Theory: The Design Problem Space. *AI Magazine*, Vol. 10, No. 1, pp. 18-36.
- Hinrichs, T.R.: 1992, *Problem Solving in Open Worlds: A Case Study in Design*. Lawrence-Erlbaum Associates, Hillsdale, NJ.
- Johnson, R.E. and Foote, B.: 1988, Designing Reusable Classes. *Journal of Object-Oriented Programming*, Vol. 1, No. 2, pp. 22-35.
- Kolodner, J.L.: 1993, *Case-Based Reasoning*. Morgan-Kaufman Publishers, Inc., San Mateo, CA.
- Kolodner, J.L. and Penberthy, T.L.: 1990, A Case-Based Approach to Creativity in Problem Solving. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. August.
- Maier, N.R.F.: 1931, Reasoning in humans II: The Solution of a problem and its appearance in consciousness. *Journal of Comparative Psychology*, vol. 12, pp. 181-94.
- Rich, C. and Waters R.C.: 1990, *The Programmer's Apprentice*. Addison-Wesley, Reading, MA.
- Selfridge, M.: 1990, Why Do Adults Tell Stories? Why Do Children Play? For the Same Reason: Re-Indexing Old Cases Under New Generalizations. *AAAI Spring Symposium on Case-Based Reasoning*, pp. 72-74.
- Tong, C.H.: 1988, *Knowledge-Based Circuit Design*. Ph.D. Thesis. Rutgers Technical Report LCSR-TR-108. Laboratory for Computer Science Research. Hill Center for the Mathematical Sciences Busch Campus, Rutgers University. May.
- Turner, S. R.: 1991, A Case-Based Model of Creativity, *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, pp. 933-937.

Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design

Janet L. Kolodner and Linda M. Wills

College of Computing

Georgia Institute of Technology

Atlanta, Georgia 30332-0280

jlk@cc.gatech.edu, linda@cc.gatech.edu

Abstract

Case-based reasoning can be used to explain many creative design processes, since much creativity stems from using old solutions in novel ways. To understand the role cases play, we conducted an exploratory study of a seven-week student creative design project. This paper discusses the observations we made and the issues that arise in understanding and modeling creative design processes. We found particularly interesting the role of imagery in reminding and in evaluating design options. This included visualization, mental simulation, gesturing, and even sound effects. An important class of issues we repeatedly encounter in our modeling efforts concerns the focus of the designer. (For example, which problem constraints should be reformulated? Which evaluative issues should be raised?) Cases help to address these focus issues.

Introduction

Designers across different domains perform many of the same creative activities, whether they are involved in designing artifacts or processes. These activities can be described by contrasting them to routine design activities. In general, routine design repeats old designs in obvious ways, adapting them by well-known and often-applied adaptation strategies. Routine design assumes a completely specified problem is given and little effort is applied to elaborating or designing a feasible specification.

The kind of design we call creative, on the other hand, includes a process of "designing the design specification" (Tong, 1988), going from an incomplete, contradictory, and underconstrained description of what needs to be designed to one with more detail, more concrete specifications, and more clearly defined constraints. Creative design also often includes a process of generating and considering several alternatives, weighing their advantages and disadvantages, and sometimes incorporating pieces of one into another. It involves using well-known design pieces in unusual ways or modifying well-known designs in un-

usual ways. Creative designers frequently engage in cross-domain transfer of abstract design ideas. They also often recognize alternative uses or functions for common design pieces (e.g., using a styrofoam cup as a boat).

Figure 1 gives a rough sketch of the main processes we hypothesize to be involved in creative design and how they interact with one another. The designer continually updates the design specification as well as a pool of design ideas under consideration. Each alternative generated is evaluated to identify its advantages and disadvantages and to check that it satisfies the constraints in the current design specification. A key part of evaluation is "trying out" the alternative (e.g., through experimentation or mental simulation). This generates a more detailed description of the alternative, including the consequences of its operation and how environmental factors affect it.

Evaluation drives both the updating of the design specification and the modification and merging of design alternatives. It raises questions of legality or desirability of features¹ of a design alternative and it detects contradictions and ambiguities in the specification. The resolution of these questions, contradictions, and ambiguities serves to refine, augment, and reformulate the design specification. On the generative side, evaluation identifies advantages and disadvantages of alternatives which often suggest interesting adaptations or ways of merging alternatives. Also, sometimes the description of a problem noticed during evaluation can be easily transformed to a description of how its solution would look.

The three processes interact opportunistically. The generative phase, guided by critiques from the evaluation phase, watches for opportunities to merge or adapt design ideas to create new alternatives. The design specification is incrementally updated as ideas are tested and flaws or desirable features become apparent.

The continual elaboration and reformulation of the problem (i.e., the design specification) derives ab-

¹The features of a design alternative are not only its structural characteristics and physical properties, but also relations between combinations of features.

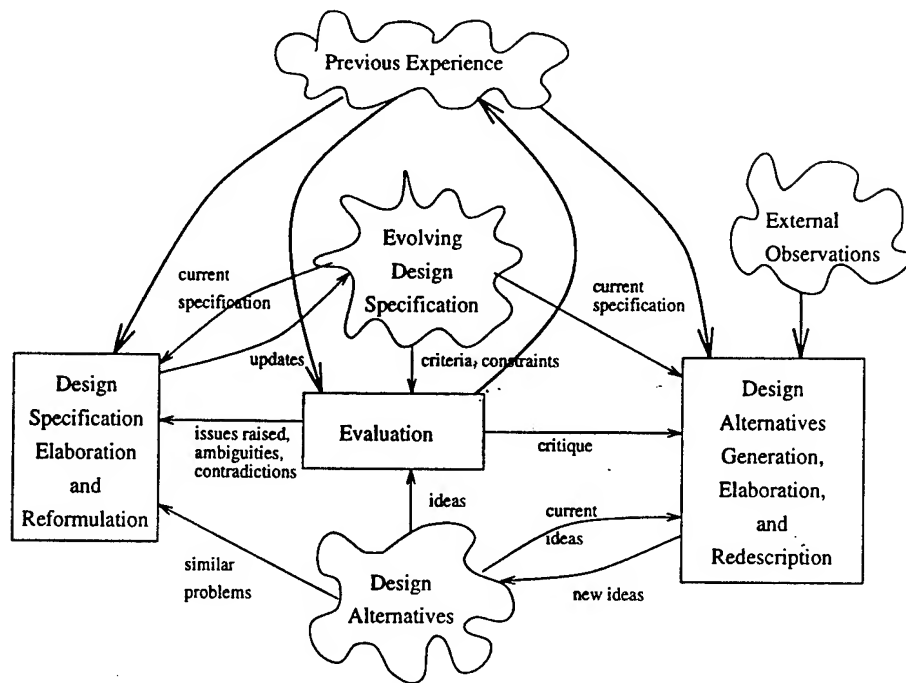


Figure 1: Rough sketch of creative design processes.

stract connections between the current problem and similar problems in other domains, facilitating cross-contextual transfer of design ideas. Continual re-description of what the solution (i.e., the evolving design) looks like primes the designer for opportunistic recognition of alternative functions of objects.

These processes rely heavily on previous design experiences and knowledge of designed artifacts. An expert designer knows of many design experiences, accumulated from personally designing artifacts, being given case studies of designs in school, and observing artifacts designed by others. Our observations and analyses lead us to propose that reminding of these experiences is crucial to generating design alternatives. When a design experience is recalled, it suggests a potential solution that can be critiqued with respect to the new problem, adapted to meet the needs of the new situation, or merged with other proposed solutions.

We believe that case-based reasoning (Kolodner, 1993) can play a large role in modeling these processes. Research in case-based reasoning has provided extensive knowledge of how to reuse solutions to old problems in new situations, how to build and search case libraries (for exploration of design alternatives), and how to merge and adapt cases. Many of the activities of creative designers can be modeled by extending routine problem solving processes that exist in current case-based systems.

Design cases provide a rich collection of details that are used in several ways in addition to generating ideas, including

- reformulating and elaborating the problem specification or proposed solutions,
- predicting the outcome of making certain design decisions,
- enabling visualization and simulation of proposed designs, and
- communicating abstract ideas in concrete terms.

What cases seem to do is to help the reasoner determine how to productively continue reasoning. The question we ask is how? How does the designer know which details to pay attention to? Which aspects of an old design can suggest problem reformulations or can fill in missing details of the specification? During problem reformulation, which constraints should be relaxed or strengthened? How are evaluative questions and criteria incrementally raised to critique the proposed design options?

We call this problem "focus." These issues are relevant in understanding what knowledge must be captured in case libraries, the form this knowledge should be in, and what types of indices are needed to allow retrieval of relevant cases. At the same time, cases help address many of these focus-related issues, particularly raising evaluation criteria and suggesting interesting, useful problem reformulations.

Example Design Episode

We concentrate primarily on an example design episode from an exploratory study we conducted of a student mechanical engineering (ME) design project.

The design task was to build a device to quickly and safely transport as many eggs as possible from one location to another. The device could be constructed from any material, but had to satisfy a set of size, weight and cost restrictions. The initial description of the problem was vague, ambiguous, and incomplete, requiring a great deal of elaboration and reformulation. One of us participated in the seven-week project as a member of a four-person team, rather than as an outside observer. Active participation in the project allowed us to become immersed in the issues the students were dealing with and to observe a great deal of the design process, including "official" as well as informal team meetings (e.g., while choosing materials at a store or while attending class).

The following is a short excerpt from a discussion early in the project concerning how to launch the eggs from the center of a child's wading pool. This excerpt was chosen because it involves a reformulation of the original problem statement. It illustrates the types of design experiences and artifacts the students typically recalled and the variety of ways they used these reminders. It also gives us some insight into the basis upon which design experiences are remembered.

1 S2: Think about how heavy eggs are....

2 S4: Yeah, we need something that's going to propel this thing. I mean it's only going this far but if you think about it, it's gotta lift up 12 inches and land over there. I've got a feeling it's really gotta propel you know [motor noise] and then just go [splat noise] with a thud.

3 S1: I've got this picture in my mind of this really dramatic missile. If it's in the water, it... it could sink and it would be like a missile coming out of a submarine. [He demonstrates, pretending his pen is a missile, makes fizzing noise] ... coming out of the water, ... splashing water out.

4 S3: That reminds me cause you see those missiles come out one at... What if we did something where we sent eggs over one at a time?

5 S3: So we could have something over there to catch them like a big pillow or something I don't know, but that way you wouldn't have to launch the whole set of them. You just launch one at a time.

6 S2: Put that down: launching individually.

[S3 records idea on post-it.]

[Unrecorded conversation while flipping tape:

7 S4: We can put them each in a tennis ball.

8 S4 mentioned ping-pong ball shooters.

8 S1 didn't know what S4 was talking about.]

8 S4: Well, they're actually little springs some of them.

8 S1: Are they?

8 S4: Yeah, you know how when we were kids we could take those things that would shoot ping-pong balls and pull them back...

8 S2: I remember those! I loved those!

8 S4: ... and shoot them? Yeah. You were a deprived child.

8 S1: Were they guns?

8 S4: Yeah.

9 S4: That's actually, hmmm. That would be about the size of an egg. If we were to send it over one at a time.

10 S2: Yeah, a lot heavier, though, the eggs.

11 Later (after this meeting), S3 visualized how the idea would work and imagined that the eggs would all end up landing at the same target spot and smash each other. So S3 thought of rotating the launch mechanism so that it throws the eggs in all directions. S3 noted one interesting consequence of this was that the eggs could be thrown all at once, each in a different direction.

12 The rotating launch reminded S3 of a recently suggested idea: "flinging motion where the device is spun around and around and then let go." This had been recorded externally on a post-it.

13 This was then adapted (generalized) from having a group of eggs at the end of the string to a single egg.

14 [Two days later, this idea was discussed further while the students were going through each idea proposed so far (recorded on post-its).]

15 S3: What I was thinking was that you could just have a pole and you could have all these strings just like a May Day dance, you know where you have all the eggs hanging from strings and you spin that and the eggs all fly out and then you just let go and then they all fly.

16 S4: Now I like... that's actually pretty interesting there, cause you could .. tie them all to something like a softball...No.

17 S4: Maybe something like... I'm trying to think of something that... What about something that's squishy?

18 S4: It's gotta have... What if it has some kind of fluid, like an orange? If you put an egg inside a hollowed out orange, half hollowed out orange, each of those little things would squash, you know inside of an orange. (I just ate an orange for lunch... I bring real-life experiences to this.)

19 S1: Well, that's the concept of a shock absorber. And the way it works is... If you just have a sealed shock. If you have... What a sealed shock would be would just be a balloon. If we had the eggs sitting on top of this big balloon and it went down, whenever the balloon squashed, there'd be pressure inside the balloon and it would jump back up again, so it would bounce.

But if you have a shock absorber that has a little seal out, whenever it... it's like a balloon w/ a little tiny hole, so whenever it hits the ground, it squashes and the air shoots out so it doesn't recoil. And an orange, whenever it's squashed, the juices would go squirting out and it wouldn't rebound.

During this design episode, the students recalled many cases, most of which are devices, some in action. Two different aspects of cases seemed to get the most attention: how a device works and what are its results (i.e., what it accomplishes, how it might fail, its pros and cons). Often, what was remembered seemed

to get embellished through a sort of mental simulation, sometimes causal (e.g., the operation of ping-pong ball shooter 8) and sometimes imagistic (e.g., the submarine launch 3, 4).

These reminders are used in many different ways.

1. They *generate* design ideas that can be re-used directly, adapted to the current situation, or merged with other design pieces. For example, tennis balls (7) and softballs (16) are recalled to be reused for the new purpose of protecting eggs.
2. They *predict* the outcome of proposed solutions. For example, the leaky shock absorber (19) is used to predict that an orange would not be a resilient egg protector. This is useful in *evaluating* proposed solutions.
3. They *communicate* ideas. For example, the May Day dance (15) is used to quickly communicate the structure of a design alternative.
4. They help simulate or visualize the behavior of a proposed design alternative. This is useful in *elaborating* both proposed solutions and vague, incomplete specifications. For example, S1's mental picture of a submarine submerging and launching a missile (3) is used to help simulate the desired behavior of the device being designed. Simulation and visualization are also key ways of collecting data to be used to *evaluate* a proposed solution. For example, the problem with the initial proposal to launch eggs individually, like a submarine does, was detected by mentally simulating the launch and realizing that all eggs end up at the same spot and could break each other (11).
5. Reminders can also lead to a complete *reformulation* of the problem. For example, remembering that submarines launch missiles one at a time (4) led to converting the problem from launching a group of eggs in a single launch to launching each egg individually in multiple launches.

Focus Issues

A number of focus-related issues come up as we examine the design episode above. We describe each here and discuss what seems to provide the necessary focus. In many instances, previous design cases themselves help direct the designer's attention.

Which cases are recalled?

Of all the design experiences each student designer has had, why are these particular ones recalled? In other words, on what basis are the cases recalled? For example, what made S1 recall a shock absorber (19) and use it to analyze the effectiveness of an orange as a structure to protect an egg?

A hallmark of a creative designer is variety. Given the same problem to solve several times, the creative designer might come up with several qualitatively different solutions. We hypothesize that this happens because on each occasion, the designer is reminded of

different cases, knowledge, or principles for solving the problem. Each time, the designer has different cues available to use for retrieval, despite the fact that the problem itself is the same. That is, the probe to memory that recalls previous designs or design knowledge includes not only the problem specification but also aspects of the context the designer is in or has been in recently.

In the given design episode, there are a variety of types of features that form the basis for reminding. Many reminders were based on a description of the *problem*, i.e., the function or behavior desired. The submarine launching a missile (3) was recalled as an example of a device that launches from water.

The ping-pong ball shooter (8) may have been recalled by looking for a device with the desired behavior of multiple launches of individual objects. In addition to the desired behavior, prominent *visual cues* may have played a role: the rounded shape and white color of the objects to be launched could have contributed to the memory probe if S4 visualized the desired behavior.

Structural cues describing the proposed solution, or structural constraints the solution should have, often remind students of an existing device that shares those features. For example, the structure of the proposed design that flings all eggs at once on strings reminded S3 of the maypole used for May Day dances (15).

Also, *background cues* can have an effect. S4 used not only structural cues (squishy, containing fluid) to recall an orange (18), but also cues from recent or current experiences (what S4 ate for lunch). Background interests provide additional cues. S1 is planning on becoming an automotive engineer and is often reminded of designs from the automobile domain, such as the shock absorber (19).

Understanding the basis for recalling design experiences is crucial to organizing a library of design cases and choosing indices to allow access to the cases. This is discussed further in the last section.

Which features of cases are examined?

Once a relevant design case is recalled, which aspects are examined? Some lead to problem reformulations or fill in missing details of the problem specification. Some are undesirable features that suggest new constraints that should be added to the problem specification to prohibit them. Some help elaborate a proposed solution. But how is the designer's attention drawn to those aspects that can do these things?

For example, there are numerous facts associated with submarines. What drew S3's attention to the fact that they launch missiles one at a time (4), as opposed to facts about how missiles are aimed at their target or about the cramped, claustrophobic interior? Focusing on this aspect led to a complete reformulation of the problem from launching a group of eggs to launching eggs individually.

When S1 used a mental picture of a submarine launching missiles (3) to elaborate the desired behavior of the mechanism being designed, why did S1 focus on sinking and then launching, but not on other aspects of the submarine's operation, such as spying on or targeting other ships using a periscope?

When S4 brought up a ping-pong shooter, first the spring mechanism responsible for shooting was considered (8). Then the weight and size of the ping-pong balls shot was considered and compared to eggs (9,10).

The reasoning goal plays a significant role in focusing attention. When S1 recalled the submarine missile launch, the team was elaborating the problem specification by describing what the mechanism should do. It was also considering the problem of launching a heavy object out of water.

In pursuing the problem elaboration goal, S1 was interested in filling in details of the behavior of the mechanism to be designed and was focused on what aspects of the submarine's launching behavior transfer over to the egg-carrying device. So S1 was drawn to coarse-grained, high-level behaviors of the submarine and missile performed when launching from water (submerging, shooting, coming out of the water). On the other hand, S3 was viewing the submarine missile launch case from the perspective of trying to borrow its solution to the launching problem. So S3's attention was drawn to the solution detail that multiple, relatively small missiles are launched one at a time. (Attention to the small nature of the missiles may have been additionally emphasized by the hand gestures S3 made in acting out the launch.)

The ping-pong ball shooter was also considered from two different viewpoints. The team considers how the gun works as part of the goal of borrowing its solution and focuses on the spring mechanism: how the spring is loaded and released. Then S4 seemed to be considering whether the gun can be reused directly. The goal of evaluating the applicability of this existing design to the current one focused S4 and S2 on the size and weight of the ping-pong balls shot, compared to eggs.

Which evaluative issues are raised?

The evaluation process checks each design option that is generated against the current design specification. It forms a critique, identifying how well the option satisfies the constraints or how badly it fails. It also notices questionable features whose desirableness or legality are unknown. In addition, a designer has goals and guidelines that are not in the initial design specification itself but whose violation or achievement can be noticed. For example, a meal planner might like meals to be easy to prepare, but may not include this in every design specification. Goel and Pirolli (1989) identify several classes of constraints that are of this nature, including domain-specific technical constraints (such as structural soundness), legislative constraints (such as building codes), common sense, pragmatic constraints

(for example, "short construction time" or personal safety), and self-imposed, personal preferences (such as "not spicy").

Not all of the evaluation criteria and problem constraints are explicit at the start of the design. They gradually surface as ideas are proposed and criticized. A key focus-related issue is: of all the evaluative issues that could be raised, why do certain ones come to mind? In the ME design project, some issues were always raised. For instance, the issue of egg safety was a primary consideration, based on the initial problem statement. Others are derived from primary goals of the designers. For example, the team was to design an egg-carrying device for at least two eggs, but one student (S2) strongly advocated that the device have a high egg-carrying capacity. This meant that S2 often brought up issues concerning how well the proposed designs accommodated the weight and space required for several eggs (1, 10).

Other evaluative issues had to be discovered as ideas were proposed. One way this sometimes occurred is that features of a proposed alternative seemed to draw attention to particular issues that might not have been considered otherwise. Some of the features are more distinctive or odd and these seem to index directly into the set of implicit criteria held by the designer. For example, during the ME design project, the students were testing how well various types of spongy material cushioned eggs when dropped from two stories. A person walked by who had done a design project which also involved protecting an egg from breaking on impact. He said he wrapped the egg in a sponge soaked in motor oil and then stuffed it in a Pringles can (a narrow cardboard cylinder in which potato chips are stacked). One of the aspects that was new about this case, compared to the ideas the students had been considering is the idea of soaking the sponge in motor oil. Focusing on the motor oil aspect reminded the students of their personal preference that the device be clean. The motor oil aspect seemed to be directly associated with the cleanliness criterion.

A second way evaluative issues are discovered is through case-based projection. Previous design cases can be used to project or derive the outcome of the current one. In the design episode, S1 recognized the similarity of the orange as a cushioning "device" to a shock absorber with a leak (19) and could predict the problem of not being able to bounce back upon impact. (S1 could also explain why, based on the causal model associated with the knowledge of shock absorbers.) This helped raise the issue of resiliency (the cushioning device must be able to bounce back) upon which to criticize the orange idea (18). Navinchandra (1991) refers to this as *criteria emergence* and he models the use of cases to raise new criteria in CYCLOPS, a landscape design program.

Which problem constraints are reformulated?

During problem reformulation, how is the designer's attention drawn to particular constraints to relax or strengthen?

Turner (1991,1993) provides an initial attempt to model the problem reformulation process, which he implemented in a program called MINSTREL. Turner proposes a case-based model of creative reasoning in which a given problem is transformed into a slightly different problem and then used as a probe to a case library. A recalled solution to the new problem is then adapted back to the original problem (using solution adaptations that are associated with the problem transformations). A set of "creativity heuristics" is used to transform the problem. Examples include generalizing a constraint (and perhaps suspending it altogether), and adapting a constraint to require a related, but slightly different outcome (e.g., injuring instead of killing).

However, MINSTREL does not address important focus questions, such as what guides the problem reformulation? Which features or constraints should be adapted? We believe that incorporating feedback from the evaluation of proposed alternatives can provide focus. Evaluation can home in on what is ambiguous or vague in the problem specification and try to take advantage of new views that result from relaxing or pushing the limits of the constraints. Also, when the need to compromise arises, conflicting constraints come into focus and the designer considers how they can be changed.

In the example episode, trying to understand how a recalled design solves a pending problem (launching a heavy projectile from the water) draws attention to a constraint that can be relaxed. S3 realized that the submarine doesn't launch one heavy object, but several relatively small missiles one at a time. This revealed a constraint in the current problem (launch all eggs at once) that could be relaxed (launch each egg one at a time).

Note that the problem of focus in reformulation is not just how does a designer know which constraint of several given constraints can productively be changed. It is also one of *revealing* the constraint in the first place. The students did not think of their problem in terms of moving a *group* of eggs in a *single* launch. They assumed the eggs would be launched all at once as a group, but this assumption was not explicit. Contrasting problems solved by previous designs with the current problem is an important way to make explicit the underlying assumptions so that the designer can decide whether the assumed constraints are essential or can be lifted.

Which problem constraints are of primary importance?

Of several solutions under consideration, one might be more appropriate than the others or several might each

contribute to a solution. Evaluative procedures must be able to evaluate each individual alternative by itself as well as in light of the others. Several focus questions arise: How is relative importance among the criteria decided? How are preferences among alternatives made?

Recalled cases seem to be important here. They suggest solutions, frameworks, design strategies and design philosophies, which can provide constraints with which to evaluate a solution and the preference criteria with which to prioritize the constraints. This also facilitates reformulating the specification, making trade-offs, and relaxing constraints. There may also be general and domain-specific strategies for setting priorities that we haven't discovered yet.

Priorities must be set flexibly, however. It is interesting that in the design episode, the reformulation of the original problem to one of launching eggs individually was proposed in response to the problem of launching a heavy object from water which would require a large launch force. However, the design at the end of the episode (flinging all eggs at once) lost this advantage of individual weaker launches, since it requires just as strong a launch force to launch all eggs as a group as it does to launch them individually, but in parallel. The designer must be able to opportunistically realize that a solution is good, even though it might not fit the original goals or address concerns that were primary earlier. If a positive aspect of a proposed solution makes a new constraint or goal explicit (e.g., "be entertaining" or "look neat") or solves some other pending problem, then the designer must be able to weaken the relative importance of the conflicting goals or constraints.

Summary: Lessons Learned and Open Issues

Our seven-week exploratory study broadened our understanding of the role cases can play in design. Not only are previous designs useful in generating design alternatives and in predicting the outcomes of proposed designs. They also aid evaluation, visualization, and simulation. These are key to performing the kinds of complex elaborations and reformulations of both solutions and problem specifications that are characteristic of creative design. In particular, previous design cases help address many focus issues that permeate these activities.

Understanding the role previous design cases play, the aspects that designers pay attention to, and on what basis cases are recalled helps determine a) the content of design cases and b) how to index them.

Case Content

From our observations of creative designers, we are starting to identify the types of information cases should contain. These include symbolic descriptions of

a device's common functions and behaviors, its structural composition, causal descriptions of how it works, and the results of its operations, how it fails, and its pros and cons. Many of these can be encoded straightforwardly in the familiar framework of typical case descriptions, which in general capture a problem, its solution, and the outcome of the solution (Kolodner, 1993). However, there are key representational issues to be solved. One is how to encode the imagistic information that seems to be a prominent part of what is recalled and reasoned about with respect to a device. Another issue is how to capture both abstract, general knowledge about devices and more specific experiences with particular devices. The design cases must be represented on several levels of abstraction, perhaps having abstract device representations associated with several more concrete cases that represent specific experiences with the device.

Indexing

The effective use of design cases depends crucially on being reminded of the appropriate cases at the right time. By investigating the types of features that reminders are based on, we are beginning to understand how to index these design cases. Useful indices include not only the function of the associated device, its behavior, and its structure, but also prominent visual, auditory and other sensory features.

In addition, non-obvious, cross-contextual reminders (which often lead to unorthodox design alternatives) are sometimes based on abstract similarities. Other reminders are based on derived or computed features rather than available ones. An important open problem is determining which kinds of derived features tend to be most useful for design, whether there is a set of derived features that is common to design across domains, and when those features get derived.

Recent studies of creative problem solving protocols (Kolodner and Penberthy, 1990) suggest that anticipatory indexing is not sufficient to fully explain retrieval. Features that were not salient at the time a case was experienced might be important for retrieval in the current situation. Drawing new, abstract connections might be a result of re-indexing cases in terms of what is now relevant or important. We hypothesize that by continually updating the design specification, designers derive abstract connections between the current problem and similar problems (possibly in other domains). These abstractions can be used to see previous cases differently.

While working on a design problem, designers often perform sensitized recognition of current design options and objects in their environment as they re-examine and re-index ideas recently brought up or experienced. For example, in the ME design project, the students were considering using a spring launching device and went to a home improvement store to choose materials. While comparing the strengths of several

springs by compressing them, they noticed that the springs bent. One student mentioned that if they were to use springs, they would have to encase the springs in collapsible tubes to prevent bending. Later, they saw a display of toilet paper holders in the store's bathroom section. They immediately recognized them as collapsible tubes which could be used to support the springs.

What is interesting is that the toilet paper holders were not immediately retrieved by the abstract index "collapsible tube." The holders had to be re-indexed under this description when they were recognized. A key to sensitized recognition is refining the description of the solution. The process of critiquing proposed ideas often yields descriptions of what an improved solution would look like: what properties it would have, what function it would provide, and what criteria it satisfies. This primes the designer to opportunistically recognize solutions in observations of the external world and in recently considered design options.

Acknowledgments. We would like to thank the members of the mechanical engineering design team, Otto Baskin, Jon Howard, and Malisa Sarntinoranont, for their invaluable cooperation. We thank Farrokh Mistree for allowing us to participate in his mechanical engineering project course and for many thought-provoking discussions throughout. We also appreciate the insightful discussions we have had with Sambasiva Bhatta, Terry Chandler, Eric Domeshek, Ashok Goel, Colin Potts, Mimi Recker, Spencer Rugaber, Terry Sargent, and Craig Zimring. This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234.

References

- Goel, V. and Pirolli, P. 1989. Motivating the Notion of Generic Design within Information Processing Theory: The Design Problem Space. *AI Magazine*, 10(1): 18-36.
- Kolodner, J.L. 1993. *Case-Based Reasoning*. Morgan-Kaufman Publishers, Inc., San Mateo, CA.
- Kolodner, J.L. and Penberthy, T.L. 1990. A Case-Based Approach to Creativity in Problem Solving. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. August.
- Navinchandra, D. 1991. *Exploration and Innovation in Design: Towards a Computational Model*. New York: Springer-Verlag.
- Tong, C.H. 1988. *Knowledge-Based Circuit Design*. Ph.D. Thesis. Rutgers Technical Report LCSR-TR-108. Laboratory for Computer Science Research. Hill Center for the Mathematical Sciences Busch Campus, Rutgers University.
- Turner, S. R. 1991. A Case-Based Model of Creativity, *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, 933-937.
- Turner, S. R.: 1993, *MINSTREL: A Computer Model of Creativity and Storytelling*, Ph.D. Thesis. University of California at Los Angeles.

Understanding Creativity: A Case-Based Approach

Janet L. Kolodner

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 894-3285
jlk@cc.gatech.edu

Abstract. Dissatisfaction with existing standard case-based reasoning (CBR) systems has prompted us to investigate how we can make these systems more creative and, more broadly, what would it mean for them to be more creative. This paper discusses three research goals: understanding creative processes better, investigating the role of cases and CBR in creative problem solving, and understanding the framework that supports this more interesting kind of case-based reasoning. In addition, it discusses methodological issues in the study of creativity and, in particular, the use of CBR as a research paradigm for exploring creativity.

This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234.

Topics in CAsE-Based Reasoning, selected papers from the First European Workshop on Case-Based Reasoning. Kaiserslautern, Germany, Springer-VErlag, 1994. Lecture Notes in Artificial Intelligence, pp. 3-20

Table of Contents

1 Background	3
1.1 An Introspective Analysis	3
1.2 Creative JULIA – our earliest computer model of creative problem solving	5
1.3 Back to real-world dieticians	6
1.4 What we've learned from these early investigations	7
2 Standard Case-Based Reasoning and Creative Problem Solving: Similarities and Differences	7
3 An Exploratory Study of Mechanical Engineers	9
3.1 The Protocols	9
3.2 The Issues	12
4 The role of cases and CBR in creative problem solving	13
4.1 Roles Cases Play	13
4.2 What are the Cases?	13
4.3 Case Content	14
4.4 Access to Cases and Other Knowledge	14
4.5 Evaluation	15
5 The nature of creativity: What have we learned?	15
5.1 Control	15
5.2 Insight, Noticing Opportunities, Serendipity	17
5.3 Where Do Ideas Come From?	18
6 Methodological Coda and Conclusions	18
7 References	19

1 Background

Several years ago, one of my students, Hong Shinn, wrote a program called JULIANA that planned meals for institutions, such as nursing homes and schools [8]. JULIANA, like any standard case-based program of its time, solved problems by remembering an old case and adapting it to fit the new situation. One of JULIANA's solutions got me thinking about creativity in a way I hadn't thought about it up to then.

When asked to plan a breakfast for a nursing-home patient with no teeth, JULIANA proposed serving orange juice, scrambled eggs (left runny), toast (ground in the blender), milk and coffee.

My reaction: Yuk. Surely, I thought, our case-based programs could do better than this.

He told me that's the way dieticians in a nursing home do it. They must plan meals for patients with a wide variety of nutritional, physical, and medical needs. For each meal, they devise a general-purpose menu and adapt it in routine and easy-to-handle ways to make it fit each specialized diet. A standard breakfast, for example, always has fruit, milk, a protein food, a carbohydrate, and a hot drink. Patients on a normal diet receive orange juice, scrambled eggs, toast and butter, milk, and coffee. A patient on a low-fat diet gets skim milk instead of milk and jelly instead of butter. A patient with no teeth gets a wetter portion of scrambled eggs and ground-up toast.

JULIANA matched what dieticians do. But I still wasn't satisfied. Not only was the program not interesting enough, but the solutions of human dieticians were also disappointing. This made me want to analyze both dieticians and our programs. What is each doing to come up with solutions? What do I like and not like about their solutions? What, if any, deficiencies in their reasoning generate mundane solutions rather than interesting ones? Could that be alleviated? How? How would I like them to solve the problem? One thing I was sure of was that I wanted them to *evaluate* whether a planned meal was sufficiently appetizing and to *continue trying* to come up with a better solution if it wasn't.

1.1 An Introspective Analysis

I began trying to answer these questions by looking at my own solution to the problem, and I attempted to analyze my own reasoning. My solution began by evaluating JULIANA's solution and considering alternatives based on that evaluation.

Evaluate: runny eggs and ground toast is boring, has lousy texture, and tastes bad.

Question: what can someone without teeth eat that tastes better?

Elaborate specifications: perhaps something liquid.

Based on this, I thought of serving "Instant Breakfast," a flavored powder with vitamins and nutrients that one mixes with milk and eats instead of breakfast. Perhaps I thought of this because it is made in the blender (as is the ground toast) and is liquid. I evaluated this suggestion.

Evaluate: easy to make, texture right, but taste is uninteresting; also, nutrition is wrong -- it should come from real foods rather than being artificially created.

Elaborate specifications: use real foods.

I now thought of serving yogurt milkshakes, made of real food, nutritious, and made in the blender. I drank them as nutritional supplements when I was pregnant. I evaluated again.

Evaluate: easy to make, texture is good: in addition to being liquid, there's a slight crunch; nutrition might still be insufficient, might need more iron or protein for older people.

Elaborate specifications: be nutritious.

The need for extra nutrition made me think of a friend who was a health nut. He used to add brewer's yeast for extra protein and vitamins to everything he ate. I also thought of the way I added extra nutrients to these milkshakes when I was pregnant -- by adding a raw egg. I evaluated again.

Evaluate: can't use raw eggs because of the possibility of salmonella poisoning; brewer's yeast is easily available.

Elaborate specifications: no raw eggs.

I adapted the yogurt milkshake recipe by adding brewer's yeast and evaluated again.

Evaluate: easy to prepare, liquid texture with slight crunch, flavorful taste, good nutrition.

I had a solution. I was finished.

There were several differences I noticed between the reasoning JULIANA did and what I did to solve this problem.

- I considered many cases and proposed many solutions, and solutions were often made up of combinations of features from several cases.
- Evaluation of proposed solutions was a primary process in my reasoning.
- Problem solving was incremental; problem solutions as well as descriptions were updated based on evaluations.
- During evaluation, I was willing to consider odd proposals for what they might contribute rather than disregarding them outright because they wouldn't work.

In short, there were two important processes I was carrying out that JULIANA did not:

1. evaluation of possibilities (leading to additional adaptation sometimes) beyond just checking to make sure given constraints were fulfilled and

2. intentional search for alternative solutions by a combination of elaborating and fleshing out a framework for a solution and searching to find something with that specification.

JULIANA was content with whatever it could come up with that fulfilled given constraints (so were the dieticians), but I was aiming toward a kind of quality or interestingness (whatever that is) that required going beyond given specifications and discovering and adding in additional ones that seemed relevant.

1.2 Creative JULIA – our earliest computer model of creative problem solving

Based on the analysis above and observations and analysis of several other people solving simple problems, one of my students built a computer program called Creative JULIA [3] that exhibited the reasoning outlined above. Creative JULIA focused on three major processes: memory search and retrieval, evaluation of alternatives, and updating a problem specification. Its task was another meal planning task – to come up with a dish that satisfied a given specification. It did this by iteratively elaborating and refining its initial specification as it recalled and evaluated possible dishes. For example, given the goal (an initial problem specification) of coming up with a dinner dish to use up some leftover white rice, it considered fried rice, decided that it didn't want Chinese food, and updated its problem specification to rule out Chinese cuisine. When it thought of making yeasted rice bread, it decided that was too time-consuming and added to its specification that the dish be easy to make. When it thought of macaroni and cheese, which it deemed appropriate if rice could be substituted for macaroni, it updated its solution specification with the framework for macaroni and cheese and substituted rice into the recipe. Since it still had rice leftover, it continued looking for something else it could use. When it thought of rice frittata, a breakfast dish, it decided that its goal of using the rice for dinner wasn't all that important, deleted that goal from its specification, and added rice frittata to its solution. Creative JULIA's reasoning framework is listed below.

- Retrieve a set of cases (initially, use the original problem specification as a guide)
- For each case:
 - Evaluate the solution proposed by the case for its applicability to the new problem
 - Evaluate the solution proposed by the case for its adaptability to the new problem
 - Based on evaluations, update the problem solution and
 - update the problem specification appropriately
- Repeat until a satisfactory solution is created or found

1.3 Back to real-world dieticians

Let me return again to the breakfast for the man with no teeth. Remember that I wanted to feed him a yogurt milkshake. Creative JULIA implemented the process I used to come up with this solution. But I want to return to this example because there's an issue I left out – some readers will say I cheated in coming up with my solution. I wasn't adhering to the same constraints a dietician in a nursing home adheres to – that the variety of meals it has to make be similar enough to each other that the kitchen staff can efficiently make all the variations. I was only trying to come up with one interesting meal.

I've addressed that problem too, and I've come up with several different methods for coming up with more interesting solutions, each of which would result in something different than the standard solution, but would adhere to efficiency constraints of the kitchen staff.

- Instead of 5 separate meal parts, use a *different meal structure* that combines parts. Sample result: french toast, yogurt with fruit and granola, milk and coffee.
- Use a *different design plan* – create a dish appropriate for nearly everyone and augment it for those it can't satisfy. Sample result: Yogurt milkshake with wheat germ in it is low-fat, low-calorie, balanced, and includes all of the food groups; it can be eaten by everyone except those allergic to particular ingredients; it can be augmented for those on a high-calorie diet with some additional dish.
- Attempt *different adaptations* – instead of replacing foods that violate some constraint with a "typical" substitute (e.g., milk with skim milk), use a more novel substitute (e.g., replace milk with low fat yogurt); instead of changing the texture of a food for someone who cannot cope with the normal texture (grinding the toast), substitute something of the right texture (oatmeal) or delete that part of the meal, or come up with a new dish that is of the right texture (e.g., yogurt milk shake with wheat germ instead of ground solids).

There are several things to notice:

1. *There are many different ways to go about solving the problem in interesting ways* – focusing on the individual problem and then making sure it fulfills broader constraints, focusing on constraints and seeing how they can be stretched and adapted, focusing on the framework from which the constraints are derived and seeing if it can be changed, and so on.
2. *There are many different qualities of answer. We don't all agree on which answers are better than others.* All might fulfill stated constraints, but which ones we believe are better depends on our own individual preferences, unstated constraints, and other things we know.

Several people, in fact, have told me that they thought grinding the toast in the blender was quite creative and that yogurt milkshakes, available everywhere now (in the US), were rather mundane. And for these people, who are not experienced in the kitchen and who have no experience planning meals

for the disabled, the original solution might indeed seem creative, while the yogurt milkshake solution, created before there were frozen yogurt stores on every corner, might seem more mundane. Quality of solution is in the eyes of the beholder. It depends very much on what one already knows and what techniques are routinely used.

3. *Simple methods of solving a problem can yield interesting solutions*, e.g., making non-standard substitutions. It is not necessary for the process to be complex in order to come up with interesting solutions.

1.4 What we've learned from these early investigations

This all happened in 1989 and 1990. JULIANA got me thinking about two things: the deficiencies in our current case-based reasoning methods and the processes involved in creative problem solving. Creative JULIA was a first attempt at dealing with those issues.

Creativity, we hypothesized, often derived from brainstorming procedures involving enumeration of the realm of possibilities (through memory search), re-description and elaboration of problem specifications (facilitating enumeration and memory search), and evaluation of proposed solutions that went beyond the stated constraints on a solution. Deriving evaluation criteria was part of the evaluation process. In addition, we identified several attitudes that seem to be taken when people are solving problems creatively: an intent to provide novelty and go beyond the usual, and comfort with and consideration of risk taking. These attitudes led to carrying out adaptation in more interesting ways – making non-standard substitutions, applying adaptation strategies in circumstances other than the ones they were meant for, and attempting to merge pieces of solutions with each other.

While case retrieval and adaptation, the primary processes of case-based reasoning were still playing a large role in the derivation of solutions, it became clear that we still didn't know everything about the framework that supports those processes and the real power they could wield.

So there were three problems I set for myself:

1. to understand better the processes of creating interesting solutions, where an interesting solution is one that goes beyond the obvious and is generated in interesting (non-obvious) ways;
2. to investigate the role of cases and case-based reasoning processes in this kind of interesting problem solving;
3. to find out more about the framework that supports this more interesting kind of case-based reasoning.

2 Standard Case-Based Reasoning and Creative Problem Solving: Similarities and Differences

The standard CBR framework has several parts [2]. First comes **situation assessment**, the process of understanding a situation well enough to begin to solve

a problem. In most of our problem solving CBR programs, we forget about this step completely, assuming that the representation we have is sufficient for solving a problem. But this is one of the places identified as an important component of creative reasoning. Respecification and elaboration of problem specifications, as Creative JULIA does, is a kind of situation assessment. In this step, we derive previously unspecified features, both concrete and abstract, and we make hypotheses about what a solution might look like.

Retrieval happens in a second step. What we found in looking at the protocols we collected as we were building Creative JULIA was that people tended to take more risks in this step when trying to be creative – their preferences seemed to change – when given a choice of several cases to use, they preferred to use the one that aimed them toward a more unusual solution.

Case manipulation and adaptation, which comes next, seems to be more interesting when people are aiming toward novelty. Non-standard substitution, discussed earlier, is one more novel way of adapting an old solution to fit a new situation. Using an adaptation strategy in a non-standard way is another. Consider, for example, how the architect Frank Lloyd Wright might have thought of including the waterfall and boulder in the design of the house Falling Water. Perhaps he applied the adaptation strategy, “incorporate obstacles,” used widely in engineering design, to architecture.

These steps are followed by **taking action** and getting results, **evaluation of results**, and **learning**. I don't want to spend time on these steps. But I do want to point out that although we give lip service to evaluation, it doesn't show up very well in the standard process. It has been seen as part of case manipulation, before a solution is executed in the world, and has been little addressed.

Recall, however, that we found that evaluation was a key in creative problem solving. Issues for evaluation are derived in the course of evaluating. One does not merely depend on constraints that have already been specified.

There are two big lessons to take from this, I think. One is that creative problem solving seems to require reflection in a way that more mundane problem solving doesn't. Second is that these processes play against each other and interact in very interesting ways. Retrieval depends on the specification of the problem, and can therefore be only as good as situation assessment allows it to be. Situation assessment depends on guidance from evaluation procedures about what might be changed in a specification. It can only be as good as evaluation allows it to be. And evaluation depends on being able to derive interesting evaluation criteria. We'll see later what that depends on. And, of course, retrieval can only be as good as the experiences that are stored in the case library.

In other words, a combination of processes, including problem elaboration, construction of alternative solutions, solution evaluation, and remembering all work in conjunction with each other to produce interesting solutions, and a set of control processes control their application. Our programs, taking the standard approach, stuck to known solutions and routine ways of adapting old solutions to come up with new ones, neglecting exploration of alternatives if something

good enough was found. This results in robust, but usually uninspired, solutions – not just in JULIANA, but in nearly every existing CBR program at that time (1989) and today.

3 An Exploratory Study of Mechanical Engineers

Our introspective studies and modeling attempts with Creative JULIA provided us with a general framework for creative problem solving, but we needed more specifics. To glean insights into how to flesh out our framework, my associate, Linda Wills, and I began carrying out another investigation. In Fall, 1992, we observed a seven-week mechanical engineering design project at Georgia Tech [4, 5]. Each team of four students designed an airline emergency egress system. As an analogy to aircraft evacuation, they had to design and build a device that quickly and safely transported as many eggs as possible as far away as possible from the aircraft, which was stuck in a pool of water. They were allowed to spend up to \$100, and the device could be constructed of any material but had to satisfy a set of constraints on weight and size. Linda was an active participant in one of the teams, audio-recording all of their conversations and keeping copies of their drawings.

3.1 The Protocols

The following is a short excerpt from a discussion early in the project about mechanisms for propelling the eggs away from their stranded vehicle. This excerpt includes a redescription of the original problem statement, it illustrates the types of design experiences and artifacts the students typically recalled and the variety of ways they used these reminders, and it provides some insight into the basis for remembering design experiences.

In this excerpt, the students were discussing how to launch their device from the water.

1 S2: Think about how heavy eggs are....
2 S4: Yeah, we need something that's going to propel this thing.
I mean it's only going this far but if you think about it, it's gotta lift up 12 inches and land over there. I've got a feeling it's really gotta propel you know [motor noise] and then just go [splat noise] with a thud.

Notice that S4 actually acted out with his pen and with sound effects how the device would behave.

3 S1: I've got this picture in my mind of this really dramatic missile. If it's in the water, it... it could sink and it would be like a missile coming out of a submarine. [He demonstrates, pretending his pen is a missile, makes fizzing noise] ... coming out of the water, ... splashing water out.

4 S3: That reminds me cause you see those missiles come out

one at... What if we did something where we sent eggs over one at a time?

5 S3: So we could have something over there to catch them like a big pillow or something I don't know, but that way you wouldn't have to launch the whole set of them. You just launch one at a time.

6 S2: Put that down: launching individually.

[S3 records idea on post-it.]

Several things should be noticed here. First, the students seem to be doing a mental simulation of egg launching. How? They begin by simulating a device they are familiar with (a submarine missile launcher) that performs the function they need to design for (launching out of water). The desired behavior is elaborated by visualizing the recalled device in action. This allows one student to notice that missiles are launched one at a time from a submarine. It allows another to imagine the landing, to realize that a hard landing would break the eggs, and to suggest a pillow to catch them.

The results of this simulation should also be noticed. The mental simulation of the submarine draws attention to a constraint that was not explicit in the original specification - that the eggs must be launched as a group. The simulation also points out that this constraint can be lifted, resulting in a reformulation of the problem to one of launching the eggs individually.

[Unrecorded conversation while flipping tape:

7 S4: We can put them each in a tennis ball.

8 S4 mentioned ping-pong ball shooters.

8 S1 didn't know what S4 was talking about.]

8 S4: Well, they're actually little springs some of them.

8 S1: Are they?

8 S4: Yeah, you know how when we were kids we could take those things that would shoot ping-pong balls and pull them back...

8 S2: I remember those! I loved those!

8 S4: ... and shoot them? Yeah. You were a deprived child.

8 S1: Were they guns?

8 S4: Yeah.

9 S4: That's actually, hmmm. That would be about the size of an egg. If we were to send it over one at a time.

10 S2: Yeah, a lot heavier, though, the eggs.

Once the idea of catching the eggs in a big pillow was suggested, focus shifted to the problem of protecting the eggs. Someone suggested putting them each in a tennis ball. But then attention shifted back to multiple individual egg launches and a ping-pong ball shooter was suggested. Ping-pong balls, like eggs, are white and round. We think ping-pong ball shooters were recalled, at least partially, on the basis of this visual cue. Next, ping-pong balls and eggs were compared, leading to the realization that eggs are heavy. The need to deal with the weight of the eggs is added to the problem specification.

Later (after this meeting), S3 visualized how one-at-a-time launch would work and imagined that the eggs would all end up landing at the same target spot and smash each other. So S3 thought of rotating the launch mechanism so that it throws the eggs in all directions. S3 noted one interesting consequence of this was that the eggs could be thrown all at once, each in a different direction. The rotating launch reminded S3 of a recently suggested idea: "flinging motion where the device is spun around and around and then let go." This had been recorded externally on a post-it. This was then adapted (generalized) from having a group of eggs at the end of the string to a single egg.

Two days later, this idea was discussed further while the students were reviewing the ideas proposed so far (recorded on post-its). In communicating the structure of the idea to the rest of the team, S3 referred to the May Day dance.

15 S3: What I was thinking was that you could just have a pole and you could have all these strings just like a May Day dance, you know where you have all the eggs hanging from strings and you spin that and the eggs all fly out and then you just let go and then they all fly.

16 S4: Now I like... that's actually pretty interesting there, cause you could .. tie them all to something like a softball...No.

17 S4: Maybe something like... I'm trying to think of something that... What about something that's squishy?

18 S4: It's gotta have... What if it has some kind of fluid, like an orange? If you put an egg inside a hollowed out orange, half hollowed out orange, each of those little things would squash, you know inside of an orange. (I just ate an orange for lunch... I bring real-life experiences to this.)

Once they considered flinging the eggs separately, the issue of cushioning came up again, this time focusing on cushioning each egg separately. The same person who had earlier suggested using a tennis ball to protect the eggs (S4) this time suggested putting each egg in a softball and tying the strings to the softball-protected eggs. He elaborated what he wanted for egg protection material, drawing on recent experiences - this time the orange he ate for lunch.

It is interesting to note that the students' priorities changed flexibly as they made tentative design decisions, backed out of them, and recognized good or bad features of the proposed designs. For example, an idea like throwing the eggs in parallel might be pursued because it is "different" or "looks cool" even though it doesn't satisfy the original goal that the launch force be small. The students seemed to opportunistically decide when a solution had the potential to be good, even though it didn't fit the original goals or address concerns that were primary earlier. If a positive aspect of a proposed solution made a new constraint or goal explicit (e.g., "be entertaining" or "look cool") or solved some other pending problem, they were willing to weaken the relative importance of the conflicting goals or constraints.

19 S1: Well, that's the concept of a shock absorber. And the way it works is... If you just have a sealed shock. If you have...

What a sealed shock would be would just be a balloon. If we had the eggs sitting on top of this big balloon and it went down, whenever the balloon squashed, there'd be pressure inside the balloon and it would jump back up again, so it would bounce.

But if you have a shock absorber that has a little seal out, whenever it... it's like a balloon w/ a little tiny hole, so whenever it hits the ground, it squashes and the air shoots out so it doesn't recoil. And an orange, whenever it's squashed, the juices would go squirting out and it wouldn't rebound.

One student knew automobiles well, and he seemed to be groping at this point for an explanation of whether the hollowed-out orange would cushion an egg well or not. A car's shock absorbers are used for cushioning, and he used his knowledge of shock absorbers to understand and predict how the orange would behave as a cushioning device. This analysis also allowed the students to refine the original constraint that their device provide effective cushioning. They now know that an answer to this question requires asking whether the egg protector has rebound and whether or not it loses its cushioning medium on impact.

3.2 The Issues

This excerpt brings up several interesting issues.

1. *What are the cases?* Our subjects remembered experiences, certainly, but they also recalled examples of devices, some in action. How are devices indexed (made accessible)? What content do device descriptions have?
2. *The role of visualization.* Sometimes visualization allows reminding. Other times it plays a role in simulation. What do cases need to have in their representations to allow them to guide visualization? What would such a representation look like?
3. *Gradual discovery of evaluative issues.* Our subjects discovered evaluative issues as they went along. The full set of evaluative issues was not known at the beginning. What is this emergence based on? Here, evaluative issues arise from comparing and contrasting proposed solutions with previous solutions and from envisioning solutions (especially how they work) and noticing problems. How else does emergence happen?
4. *The varied roles of cases.* Cases play several significant roles, not only for suggesting solutions, but also for communicating (explaining), providing a basis for simulation, predicting the outcome of proposed solutions, and elaborating vague, incomplete specifications.
5. *Control.* So many things that could be reasoned about next; how is one chosen?
6. *The role of serendipity.* What one is able to do depends so much on what one is able to remember, and that depends as much on what else is available in the environment as it does on what one is actually working on.

I want to spend time on two big issues – first, the role of cases and case-based reasoning in creative problem solving, and second, what we now know about creativity.

4 The role of cases and CBR in creative problem solving

4.1 Roles Cases Play

We already know that cases play a major role in *generating ideas* that are re-used directly, adapted to the current situation, or merged with other design pieces [2]. For example, in our excerpt, tennis balls (7) and softballs (16) are recalled to be reused for the new purpose of protecting eggs. We also know that cases are useful in *predicting outcomes* of proposed solutions. For example, the shock absorber (19) is used to predict the behavior of an orange used as an egg protector. This is useful in *evaluating proposed solutions*.

This study shows us that, in addition, cases *facilitate the communication* of ideas. For example, the May Day dance (15) is used to quickly communicate the structure of a design proposal.

They also aid *visualization* and provide the basis for *simulation*. This is useful in *elaborating* both proposed solutions and vague, incomplete specifications. For example, S1's mental picture of a submarine submerging and launching a missile (3) is used to help simulate the desired behavior of the device being designed. This also led to a *reformulation* of the problem: from launching a group of eggs in a single launch to launching each egg individually in multiple launches. Simulation and visualization are also key ways of collecting data to be used to *evaluate* a proposed solution. For example, the problem with the initial proposal to launch eggs individually, like a submarine does, was detected by mentally simulating the launch and realizing that all eggs end up at the same spot and could break each other (11).

4.2 What are the Cases?

This brings us to another interesting question: what are the cases anyway? In the design situations we are investigating, there are several kinds of cases. Some are *experiences with designs in the same domain* (e.g., earlier high school egg drop competitions in which egg protection mechanisms were designed and tested against each other). Some are *experiences in experimenting with proposed designs* (e.g., mock-ups or prototypes of partial designs). Other cases are *experiences with a common device*, usually in some phase of its lifecycle, focusing on some slice of behavior or functionality – how it behaves in certain situations, what it accomplishes, how it fails (e.g., the ping-pong ball shooter). Sometimes the behavior or functionality considered is not its primary one. For example, in an excerpt not shown here, a yo-yo was suggested as a means of slowly lowering an object (the egg carrier) by converting potential energy to angular (versus linear) kinetic energy. Another time a rubber raft was suggested as something that could have a hole punched into it so that it could move something and provide cushioning. The raft with a hole in it and eggs riding in its dimples was suggested as a launch and transport device.

But specific experiences with devices are not all that is remembered. Sometimes, common objects and devices are recalled and reused directly, often for

a new purpose. For example, the tennis ball, softball, and orange fall into this category in our excerpt.

4.3 Case Content

Our studies of creative problem solving have also helped identify additional types of information cases should contain. We already knew from previous CBR work that cases should contain symbolic physical and causal descriptions of a problem, its solution or response, and the outcome of the solution [2]. For example, design cases should encode symbolic descriptions of a device's common functions and behaviors, its structural composition, causal descriptions of how it works, and the results of its operations, how it fails, and its pros and cons.

We also know now the importance of the visual component of cases. In this study, imagistic information seemed to be a prominent part of what was recalled and reasoned about. The visual component embellishes physical descriptions and enhances a reasoner's ability to simulate. Some reasoning observed in our design study could not have been done easily without visual representations being available.

4.4 Access to Cases and Other Knowledge

Access to cases and other knowledge is also something we learned about. We already knew some of the influences on retrieval and accessibility:

- the closeness of the problem being addressed to other problems experienced by the reasoner,
- the variety of experiences a reasoner has had and the ability to notice connections between them,
- the reasoner's depth of knowledge of a domain (this effects the ability to index accurately), and
- recent reasoning context.

We've now discovered others. Cues available at retrieval time and contributing to reminding also come from recent experiences, environmental cues, and personal interests.

For example, in the example excerpt, S4 used not only structural cues (squishy, containing fluid) to recall an orange (18), but also cues from recent or current experiences (what S4 ate for lunch). S1 is planning on becoming an automotive engineer and knows that domain well. He was often reminded of designs from the automobile domain, such as the shock absorber (19).

Our subjects cued on perceptual properties - static ones (e.g., color and shape), as well as dynamic ones (e.g., motion trajectories). By themselves, these cues aren't sufficient for retrieval, but in conjunction with the more primary cues (those describing the problem situation), they help focus the retrieval process, determining preferences for what is retrieved.

4.5 Evaluation

We also learned about evaluation. Evaluation affects both the solution in progress and the problem specification. The issues raised point out opportunities to augment or refine the design specification. The pros and cons that are described in the critique of a design idea are used by the idea exploration process to compare the idea to other options, merge and adapt alternatives, and improve promising ideas.

Our work on Creative JULIA had already allowed us to discover four classes of evaluative questions that designers routinely raise. *Constraint-related* questions ask how well an alternative solution fits the current design specification. *Function-directed* questions evaluate how well the required function is achieved. For example, the purpose of recipe creation is to create something that can be eaten, so evaluative questions arise from the concept of edibility, focusing on the taste and appeal of a dish to see if it is edible. Some evaluation questions are *derivation-driven*. Previous solutions provide a rich and important source of issues if the considerations taken into account in creating them are saved. Finally, some questions are *outcome-related*. Previous design cases can be used to project or derive the outcome of the current one.

This new study gave us a better understanding of the ways in which these issues are incrementally raised or "revealed" as the evaluation proceeds. Navinchandra [6] calls this *criteria emergence* and shows an example of how it can arise from case-based projection. In addition to criteria or evaluative issues, constraints [7], preferences, and relative priorities among them also gradually emerge. This type of evaluation is a key driving force within creative design, feeding back to situation assessment as well as guiding case manipulation. Cases play a major role in evaluation and incrementally raising new issues.

5 The nature of creativity: What have we learned?

So far, this paper has discussed what we have found out about interesting problem solving and about what we've learned about CBR processes. We're at a point now where we can begin to answer some important questions about creativity itself.

1. The problem of control: what step comes next? How can creative exploration be guided?
2. What is the nature of insight? How are opportunities recognize?
3. Where do ideas come from?

5.1 Control

The designers we observed did not follow a rigid, methodical plan detailing what to do next. Rather, they moved fluidly between various problem pieces and design processes (idea generation, adaptation, critiquing, problem refinement, elaboration, redefinition, etc.) in a flexible and highly opportunistic manner.

Sequential composition of the basic CBR processes seems far too restrictive. Rather, these processes seem to be highly intertwined and to interact in interesting ways [9]. For example, problem elaboration and redescription tactics specify contexts for search that retrieval processes use, while evaluation of recalled or adapted alternatives feeds information back to these situation assessment tactics, resulting in even better contexts for search. In some cases, what suggests a particular problem refinement or redescription results from trying to confirm the legality of a proposed solution during evaluation, and finding a loophole or ambiguity in the current problem specification. In addition, comparing and contrasting a proposed solution with other proposals during assimilation can bring new evaluative issues into focus.

CBR systems need to break out of their typically rigid control structure and allow more interaction and opportunism among processes. This requires making strategic control mechanisms explicit, so they can be easily modified, reasoned about, extended, and learned. More research needs to be directed at identifying and capturing the types of strategic control heuristics designers use.

Our study has revealed several. For example, in strategically trying to choose which piece of a problem to work on next, designers concentrate on parts of a problem that are still open after a previous solution is retrieved and partially reused. They make tentative hypothetical commitments for open design decisions to simplify the problem or make it more contextualized. They follow opportunities, dynamically changing their priority structure among constraints if very interesting or unexpectedly good solutions are stumbled upon. They put problems on the back burner when an impasse is reached or if the problem involves issues that are too detailed for the current stage of design. The serendipitous appearance of a solution may bring the problem back up.

Our designers employed a variety of strategic control heuristics, some of which are opportunistic. For example, when an alternative was proposed that satisfied some desired criteria extremely well compared to the other alternatives, they directed their efforts toward elaborating that alternative, optimistically suspending criticism or discounting the importance of criteria or constraints that were not satisfied as well. Sometimes this led to reformulation of the problem as constraints were relaxed or placed at a lower priority.

A key part of being able to take advantage of such opportunities was being able to judge whether progress was being made along a certain line of attack and to choose which ideas were more promising than others or more likely to lead to something unusual and novel.

Some strategic control heuristics were more deliberate, based on reflection. For example, one heuristic our designers used was to try quick, easy adaptations of a proposed solution first before stepping back and reformulating the problem or relaxing constraints. Other deliberate heuristics attempted to make non-standard substitutions, apply adaptation strategies in circumstances other than the ones they were meant for, and merge pieces of separate solutions with each other in non-obvious ways.

In many cases, the processes that are composed together leading to a novel

redescribe a new problem in a way that is similar to something that we've seen before.

5.3 Where Do Ideas Come From?

Finally, I want to deal with where ideas come from – the big creativity question. Case-based reasoning itself gives us a variety of answers:

- remembering
- adapting known ideas
- reinterpreting an idea
- specializing an abstract idea (making it concrete)
- elaborating known ideas
- merging pieces of ideas with each other
- explaining
- evaluating

We might call these tactics for creating ideas. Our investigation shows that the intent (or strategy) of going after novelty provides novel ways of carrying out tactics. When remembering, novel solutions are preferred. When adapting known ideas, non-standard substitutions and out-of-context use of adaptation heuristics are common. Ideas are reinterpreted by relaxing constraints, decomposing differently than usual, and redescribing from a different point of view. Redescribing may be based on the reasoner's personal interests or areas of expertise. Ideas are elaborated by visualizing the details. Ideas are made concrete or specialized, through non-standard substitution and visualizing the abstract. Evaluation is carried out after novel bases for evaluation have been derived.

Another strategy in play at the same time also helps with going after the novel: Don't evaluate too much too soon.

What is particularly interesting about this explanation of where ideas come from is that it shows the complex interactions between these various strategies and tactics – the same interactions we've pointed out earlier. The strategies and tactics work in conjunction with each other – elaboration and reinterpretation tactics specifying contexts for search that retrieval processes use. Evaluation feeds its results back to these situation assessment tactics, which in turn derive even better contexts for search.

6 Methodological Coda and Conclusions

Before ending, I want to go back to another pet issue of mine, a bit removed from what has been discussed so far – methodology. Most people view case-based reasoning as an approach to building intelligent systems; some people see it as an approach to building human-machine systems that interact with people in natural ways. But few people see case-based reasoning as a research paradigm for addressing new problems, as providing tools for investigation that go beyond what other research paradigms can provide.

Maggie Boden [1] wrote a wonderful book about creativity, which I heartily recommend. She shows far better than I do how creativity emerges from the complex interactions of processes. She gives wonderful examples of the experiences of creative people and of computer programs that begin to show creativity, analyzing where the creativity comes from. She presents marvelous analogies to introduce and help readers understand topics.

But as good as she is at analysis, Boden does not have the paradigmatic tools for finding detailed computational answers to how the different processes work. Her paradigm can analyze and critique, discovering that one's experiences and what one knows play a large role in creative thought processes. It can explain some processes in terms of rules and heuristics. But more important than rules in addressing creativity are the ways in which experiences are used: how they are remembered and what makes them memorable; how they are manipulated, adapted, merged together; and how they help with evaluating proposed solutions, and so on.

Case-based reasoning provides tools for investigating the role cases play in problem solving. What we knew already helped us to address these issues; what we found out in this investigation is helping us understand case-related processes and issues more deeply than we've been able to before.

What is creativity? Certainly in this short paper I haven't been able to tell you all about how all its processes work and are interconnected. But by taking a case-based approach to studying these processes, their control, and the representations that underlie them, we can now specify them in ways that nobody has been able to do before. We can discuss the input, output, and processing of elaboration, evaluation, merging, adaptation, remembering, simulating, and more, and we can at least conjecture about the heuristics that control their application. We still have more work to do, clearly, in fully defining our theory. But through case-based reasoning, we've been able to get a handle on what needs to be addressed, and through addressing these issues, we're beginning to have a far more sophisticated notion of the power of case-based reasoning.

7 References

References

1. Boden, M. 1992. *The Creative Mind: Myths and Mechanisms*. New York, NY: Basic Books.
2. Kolodner, J.L. 1993. *Case-Based Reasoning*. Morgan-Kaufman Publishers, Inc., San Mateo, CA.
3. Kolodner, J.L. and Penberthy, T.L. 1990, A Case-Based Approach to Creativity in Problem Solving. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. August.
4. Kolodner, J.L. and Wills, L.M. 1993a. Case-Based Creative Design, *AAAI Spring Symposium on AI and Creativity*. Stanford, CA. Reprinted in *AISB Quarterly*, Autumn, 1993, No. 85. pp. 50-57.

5. Kolodner, J.L. and Wills, L.M. 1993b. Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design. *AAAI Case-Based Reasoning Workshop*. Washington, D.C., pp. 19-25.
6. Navinchandra, D. 1991. *Exploration and Innovation in Design: Towards a Computational Model*. New York: Springer-Verlag.
7. Prabhakar, S. and Goel, A. 1992. Performance-Driven Creativity in Design: Constraint Discovery, Model Revision, and Case Composition. In *Proceedings of the Second International Conference on Computational Models of Creative Design*. Heron Island, Australia.
8. Shinn, H. 1988. The role of mapping in analogical transfer. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*. Northvale, NJ: Erlbaum.
9. Wills, L.M. and Kolodner, J.L. 1994a. Towards More Creative Case-Based Design Systems, to appear in the *Twelfth National Conference on Artificial Intelligence (AAAI-94)*. Seattle, WA.
10. Wills, L.M. and Kolodner, J.L. 1994b. Explaining Serendipitous Recognition in Design, to appear in *The Sixteenth Annual Conference of the Cognitive Science Society*. Atlanta, GA.

Explaining Serendipitous Recognition in Design

Linda M. Wills
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
linda@cc.gatech.edu

Janet L. Kolodner
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
jlk@cc.gatech.edu

Abstract

Creative designers often see solutions to pending design problems in the everyday objects surrounding them. This can often lead to innovation and insight, sometimes revealing new functions and purposes for common design pieces in the process. We are interested in modeling serendipitous recognition of solutions to pending problems in the context of creative mechanical design. This paper characterizes this ability, analyzing observations we have made of it, and placing it in the context of other forms of recognition. We propose a computational model to capture and explore serendipitous recognition which is based on ideas from reconstructive dynamic memory and situation assessment in case-based reasoning.

Introduction

When creative designers have been deeply engaged in a problem, they are often able to recognize solutions to it in their environment, even if they are not actively working on the problem at the time. The solutions recognized may be objects, behaviors, processes, techniques – anything that they observe or examine. This can often lead to key insights, sometimes revealing new functions or purposes for common design pieces. This paper takes a step toward understanding this ability by focusing on the serendipitous recognition of *objects* as solutions to pending (possibly suspended) problems during creative mechanical design.

Serendipity often plays a significant role in creativity. Its accidental nature may seem to put it out of reach for creating computer-based design systems that can take advantage of serendipity, or for supporting people in serendipitous discovery of solutions. However, we believe that being in the right place at the right time is not the difficult part; we can put a designer (human or computer-based) into a rich environment of stimuli where “accidents” will happen. The creativity comes in the preparation that allows recognition of a solution when it is present (Seifert, et al., 1994). This requires becoming immersed in the problem, redescribing it and viewing it from multiple perspectives, considering, comparing, and critiquing several options, so that when a relevant solution is spotted, the way it fits into the problem can be immediately discerned.

The following are typical examples of serendipitous recognition. They occurred during a mechanical engineering (ME) design project we observed in which a team of four students were to design and build a device to quickly and safely transport as many eggs as possible from one location to another. (The first author participated as a member of the team in order to become immersed in the design issues and observe as much of the design process as possible in a natural setting.)

Our first example (“bending springs”) occurred while our designers were considering using a spring launching device and went to a home improvement store to look into materials. While comparing the strengths of several springs by compressing them, they noticed that the springs tended to bend. One designer wrapped a hand around the spring to hold it straight as it was compressed and said the springs would each need to be enclosed in a tube to keep them from bending. Another added that the tube would need to be collapsible (to compress with the spring). The designers could not think of an existing collapsible tube and did not want to build one due to time pressure. They gave up on the springs and started thinking about egg protection. During their search for protection material, they walked through the bathroom section of the store, where they saw a display of toilet paper holders. They immediately recognized them as collapsible tubes which could be used to support the springs.

The second example (“weighted rubberbands”) occurred later in the design, after the designers had decided to use a cylindrical egg carrier with a spring launch device. They were working on a homework assignment which involved formalizing the tradeoffs between egg capacity, weight, and launch force as a multi-goal optimization problem to help them make the best choice for these variables. They were having trouble and were easily distracted from the problem. One distraction came from a designer who described a trick involving a moving cylinder (coffee can) she had seen the night before on a children’s science TV show (Beakman’s World). The episode showed how to make a coffee can that rolled back to you when you rolled it away. Batteries were taped as weights to rubberbands, strung through the center of the can. The weights caused the rubberbands to become wound up as the can rolled. As the rubberbands unwound, they caused the can to roll back to the starting location. The designers discussed whether this could be modified for use in their design (e.g., wind the rubberbands up and let their unwinding launch the device). However, they rejected the design for adding too much weight, since the design task had strict weight restrictions. Then they went back to their homework. Suddenly, one designer suggested using the *eggs* as weights on the rubberbands. This alleviated the weight problem because the weight of the eggs did not count into the restricted weight constraint.

There are three intriguing characteristics of this type of recognition. First, designers are able to recognize solutions to problems that have already been suspended. We call this *serendipitous recognition*.

Second, they are often able to recognize objects as solutions, even though this requires the object to play a role or provide a function different from its usual role or function. In our bending springs example, the toilet paper holder is not used to hold a paper roll, but to keep a spring from bending upon compression. In the weighted rubberbands example, the eggs not only play their usual role of being cargo/passengers in the egg carrier, but also are used in a nonstandard way to provide weight to make the device move. This type of recognition requires overcoming functional fixedness (Maier, 1931, 1970; Duncker, 1945).

Third, the solutions recognized are not always standard solutions to the problem. In the bending springs example, the problem was new to the designers and did not have standard solutions. In the weighted rubberbands example, the designers recognize a solution that is different from the existing standard solutions to the problem. In general, if a problem has standard solutions (or ones that are apparently applicable), fixation on these standard solutions must be overcome.

These three characteristics make this type of recognition more difficult to model than types previously studied. Since the solutions are nonstandard, it is not addressed by object recognition work (e.g., (Grimson, 1990)) which typically searches a scene for particular models of existing standard solutions to problems. The type of recognition we are interested in requires models of objects to be constructed on the fly. Jordan and Shrager (1991) model how people select objects for a nonstandard use, based on which object's physical properties relate best to the desired function. However, they do not address how the salient properties are derived or determined.

The opportunistic nature of serendipitous recognition makes research on predictive encoding (Patalano, Seifert, & Hammond, 1993) and opportunism (Hammond, 1989; Mueller, 1990) relevant to our research. However, they address opportunism in planning situations in which there are standard solutions to the problems at hand. The opportunities detected are those to fulfill suspended goals by recognizing that a standard plan for them can now be resumed. Our recognition involves recognizing objects that are not the standard solutions to the pending problems.

Some important issues serendipitous recognition raises are: To what extent does problem and solution need to be described and elaborated for the recognition to take place? How is this description created? How is the opportunity to solve a suspended problem noticed? What allows the relevant problem context to be recalled when its solution is seen?

This paper presents our analysis of the processes involved in serendipitous recognition of nonstandard solutions. Our hypothesis is that recognition arises from interactions between two processes: *problem evolution* and *assimilation* of proposed ideas into memory. We propose a computational model of serendipitous recognition based on this hypothesis. Our model draws on and extends ideas from reconstructive dynamic memory (Schank, 1982; Kolodner, 1983) and case-based reasoning (Kolodner, 1993), particularly situation assessment and evaluation processes.

Evolution and Assimilation

A key activity of designers is to understand, refine, elaborate, and re-define the problem. They view the problem from multi-

ple perspectives and redescribe it in more familiar terms. This process, which we call *problem evolution*, reveals constraints, features, and properties to look for in proposed solutions. As design alternatives are proposed and explored, they are *assimilated* into memory; they are compared and organized, based on the criteria and features the designer has become attuned to through problem evolution. Some serendipitous recognition may arise from assimilation activities.

Problem evolution is driven primarily by the *evaluation* of proposed ideas, which, in addition to revealing flaws in the specification (such as contradictions and ambiguities), generates new criteria, constraints, and preferences that go beyond those given in the original statement of the problem. Assimilation itself can play a significant role in evaluation by drawing attention to features of proposed ideas that are unusual or particularly good or bad compared to other proposed ideas. This in turn can trigger a complete problem reformulation. So, while problem evolution can "set up" the reasoner to recognize solutions when they are stumbled upon, the recognition itself can sometimes actually trigger a problem redescription. This occurred in the weighted rubberbands example, which we analyze in depth in the next section.

These interacting processes fit well into a model of dynamic memory and case-based reasoning. A key idea underlying *dynamic memory* is that remembering, understanding, and learning are all inextricably intertwined. The ability to determine where something fits in with what we already know (understanding) is a key part of being able to assimilate objects in our environment into our problem solving. This may involve a useful reinterpretation of something already in memory and can result in a new way of indexing it in memory.

The process of elaborating and redescribing the problem specification corresponds closely to the process of *situation assessment* in case-based reasoning: redescribing a problem situation in the vocabulary of problems solved in the past (i.e., the indexing vocabulary of the reasoner's memory). These processes facilitate retrieval in compensating for the fact that we may not be able to anticipate how we might want to use some piece of knowledge when we enter it in memory. Situation assessment aligns the vocabularies of the current situation with that of previous problems we have encountered. Also, by providing several different ways of describing a problem and what would count as a solution, it allows entities to be reinterpreted in the context of the problem and serendipitously recognized as relevant to solving it.

Research into situation assessment and problem reformulation (e.g., in CASEY (Koton, 1988), CYRUS (Kolodner, 1983), MINSTREL (Turner, 1994), BRAINSTORMER (Jones, 1992), and STRATA (Lowry, 1987)), show different ways this can be done. Also, Sycara and Navinchandra (1989) have identified several index transformation techniques relevant to case-based design. We are building on and extending these ideas, exploring in particular how they can be synergistically integrated with evaluation, retrieval, adaptation, and assimilation processes.

Analysis of Examples

Bending Springs

For the designers to recognize the toilet paper holder as a solution to the problem of bending springs, they needed to create a

description of what solutions to this problem would look like. This description evolved as they thought about the problem, proposed solutions to it, and critiqued these solutions. Here is a closer look at how this description evolved.

As the designers were comparing the strengths of various springs, one designer compressed the springs between a thumb and forefinger and noticed that the spring bent, imposed lateral forces at the endpoints, and a variable longitudinal force. This was judged to be a problem. We hypothesize that this judgment was made based on a violation of the designers' expectations about how the spring would behave and reasoning about the consequences of the actual behavior in the context of their proposed design.

One designer wrapped a hand around the spring to hold it straight as it was compressed and said the springs would each need to be enclosed in a tube to keep them from bending. Wrapping a hand around something to make its shape conform to what you want is a standard technique. The subsequent tube proposal can be the result of a memory retrieval based on structural shape similarity.

Another designer added that the tube would need to be collapsible (to compress with the spring). This adaptation may have been suggested as a result of noticing that the hand wrapped around the spring hindered the compression of the spring because it was too longitudinally rigid. This could be fixed by making the tube longitudinally flexible (collapsible).

Weighting Rubberbands

In the bending springs example, the designers derived a concrete description of what they needed, which primed them to recognize it when they saw it. In our second example (the problem of weighting the rubberbands inside the cylindrical egg carrier), the description of what was needed did not fully evolve before the recognition (of eggs as weights) occurred. Rather, the recognition itself helped to redescribe the problem.

The original design, in which batteries were used as weights, was rejected because the batteries would add weight to the device. Their problem description – “find something to act as a weight without adding weight” – was overconstrained. So the problem was abandoned.

They went back to the optimization problem they were given for homework, which involved thinking about the trade-off between launch force and egg capacity (the more eggs, the more force required because the eggs would increase the weight). Considering the eggs as providing weight prompted one designer to suggest the clever idea of using the eggs as weights on the rubberbands.

Our hypothesis is that the designer saw the relevance of the eggs to the weighting rubberbands problem due to its weight property, which they were focusing on in the homework. The eggs were different than the previously proposed solution (batteries) with respect to the weight property, since egg weight does not count into the total weight limit of the device. (The problem statement explicitly restricted the “weight of the device (not including the eggs).”) This difference was noticed and seen to be a key advantage. It generated a refinement of the problem description. Instead of “provide weight, without adding weight,” it became “provide weight, without adding weight that counts toward the weight limit.” It is only by bringing the eggs into focus and re-interpreting them from

the point of view of their weight that this other problem description was created. If the designers had redescribed the problem in this way to begin with, they might have immediately recalled eggs as a solution to this problem. However, the redescription seems to have been the result of the recognition rather than a prerequisite of it.

Modeling Implications

A number of interesting issues arise in considering how to model the problem evolution and assimilation occurring in these two examples.

1. Experimentation plays an important role in problem evolution. Its results are used in evaluating proposed designs and in suggesting solutions and adaptations (e.g., wrapping hand led to tube suggestion; compression hindered by wrapped hand led to the “collapsible” adaptation). Simulating or actually performing this type of experimentation (e.g., with a robot) is, of course, outside the scope of our modeling efforts. But we can provide the results of experimentation as input to our computational model.

2. As proposed solutions are generated and explored (e.g., by collecting experimental data about them), an evaluation process notices their problems (e.g., constraint or expectation violations) or good features. New evaluative issues emerge that go beyond the stated constraints on the problem. Navinchandra (1991) calls this *criteria emergence*. In addition, constraints in general (Bhatta, Goel, & Prabhakar, 1994) and relative priorities among them, also gradually emerge. This emergence is a major part of problem evolution.

3. The designers in our examples generated descriptions that allowed immediate recognition of satisficing solutions to the pending problems. They were concrete enough to be easily recognizable (for minimum inference at recognition-time) but abstract enough to be satisfied by a variety of different objects. They referred to both *structural* (cylindrical shape, length, radius, or weight) and *behavioral* (length varies) properties of objects. Matching these properties against objects under consideration sometimes requires drawing on knowledge about the object (e.g., what configurations or shapes it can take, whether it can stand on end, or whether its length can vary). This in turn requires that the object being viewed has been recognized as its usual identity (e.g., toilet paper roll holder) so that the associated structural, behavioral, and functional knowledge of the object can be matched against the evolved description of what is needed. In our computational model, the input from the environment is augmented by the equivalent of results of standard object recognition techniques, so that an object under view has both its current imagistic features and standard knowledge about its assumed structure, behavior, and function.

Computational Model

Based on our analysis of these examples and others from our exploratory study of the ME design project, we are constructing a computational model of serendipitous recognition. We are implementing this model in a system called IMPROVISER (Invention Modeled by Problem Redescription, Observation, and evaluation. Interacting SERendipitously).

IMPROVISER's proposed architecture has a problem evolution component which is modeled using situation assessment

procedures co-routined with evaluation techniques. The output of this component (i.e., the evolving specification) feeds into memory retrieval and update processes.¹ Retrieval interfaces with a library of design cases which models, in part, long-term memory. The specification is used as a probe to recall relevant design cases (for evaluation, elaboration, etc.).

Memory update is the complement of retrieval. It accumulates design alternatives proposed (i.e., those retrieved, elaborated, or viewed directly in the external environment) into a pool of design alternatives under consideration. It organizes and compares the alternatives with respect to each other, along the dimensions relevant to the problem specification. This is used to model the assimilation process. Recognition of a solution results when an alternative is stored that is a close match to the desired solution.

The data structure holding the set of design alternatives forms an extension of the long-term memory. We call this extension the *problem context*. It contains the set of explored design alternatives and the relevant set of descriptors from the specification. As the problem specification evolves, the focus changes on the relevant descriptors to be used for organizing alternatives in the memory (e.g., shape, construction cost, personal safety). For complex problems, with many subproblems, there are several subproblem contexts, which might overlap, depending on interactions between subproblems. When an alternative is entered into memory, it is interpreted with respect to the descriptors in the subproblem contexts to find the best place(s) to store the alternative.

The various subproblem contexts can be seen as dynamically constructed models of desired solutions, built during problem evolution. Recognition of instances of these models occurs as alternatives are entered into the most appropriate contexts through standard memory update techniques.

A key part of this assimilation process is noticing "interesting" similarities or differences between alternatives being added to some context. For instance, in the weighted rubberbands example, eggs were noticed to be different than previous proposals to the moving cylinder problem with respect to the weight property – one of the descriptors in the problem contexts for both the homework assignment and the moving cylinder problem. Knowing that the weight of eggs is exempt from the weight restriction makes the noticed difference interesting; it is directly related to the conflicting weight constraint, suggesting that this constraint should be re-evaluated with the eggs acting as weights. The success of this evaluation subsequently causes the weight constraint to be refined.

A set of monitoring procedures are associated with each process and watch for opportunities for further processing to occur. The opportunities noticed are placed on an agenda, maintained and accessed by strategic control heuristics.

A Proposed Scenario

This section gives a scenario of how IMPROVISER will model our bending springs example, once fully implemented. IMPROVISER starts with a partial specification which includes specifications for each subproblem in the current partitioning

of the problem (launching, moving, stopping, and protecting the eggs). The launch subproblem specification contains a partial specification for a spring launch mechanism. (In the following, "??" denotes incompleteness due to pending decisions; "..." denotes parts of the specification not shown.)

```
<Spec:
Subproblem: Launch
  Parts: Spring, Base
  Attached(Spring, Base, <position>)
  Spring:
    k: ?? ;; force constant
    x: ?? ;; spring displacement
  Launch-Force: (- (* k x)) ;; Hooke's law
...
Subproblem: Protect-Eggs ...
Subproblem: Transport ...
Subproblem: Stop ...
...>
```

This specifies that the launch mechanism should consist of two parts, attached to each other in a particular configuration (given in <position>). There is a pending decision as to the choice of spring strength (k) and how much it should be compressed (x) to achieve a certain launch force. (There are several other constraints involving the launch force which are not shown, such as constraints relating striking distance and launch force or relating the type and amount of egg protection material with the launch force it must cushion.)

IMPROVISER chooses to work on the pending decision concerning spring strength.² It asks an oracle to perform a trial-and-error experiment for it to help choose a spring from a set of springs. The oracle compares several springs and feeds back experimental data to IMPROVISER. The data is augmented with causal information about how the data resulted from the properties of the partial design. The oracle reports that the spring bends, which causes it to exert a weak, variable force in the direction of its axis and additional forces of variable magnitude and direction.

In general, there may be many results reported by the oracle. An important issue we are dealing with is how IMPROVISER directs its focus of attention to particular pieces of experimental data. In this case, the focus was on the launch mechanism and its force, so it is natural that IMPROVISER would attend to facts related to forces from the spring.

IMPROVISER notices that these results are not what is expected. It derives their consequences based on causal connections between constraints in the specification. The force along the spring's axis is weaker than the ideal launch force ($F = -kx$), which will make the device move slower and not as far as desired. The additional forces in various directions may cause an inconsistent, unpredictable motion.

IMPROVISER's evaluation monitors detect the negative consequences and update the specification to prohibit their causes, specifying that the spring in the launch mechanism must stay straight. In general, discovering the constraints to add to the specification which will require or prohibit some observed feature of a device involves reasoning based on a causal model of the device (Bhatta, Goel, & Prabhakar, 1994).

¹This section sketches only the main data flow relationships between these processes. IMPROVISER has a flexible, opportunistic blackboard-style architecture which is guided by explicit strategic control mechanisms.

²How this decision is chosen, and, in general, how IMPROVISER's sequence of steps is controlled is an interesting modeling issue. However, it is not the subject of this paper, but see (Kolodner & Wills, 1993; Wills & Kolodner, 1994).

A standard method of forcing a small object to maintain a desired shape is by holding it in that shape with your hand. IMPROVISER asks the oracle to do this and the oracle reports that this causes the spring to stay straight.

IMPROVISER evaluates the results, judges the spring staying straight as positive, and updates the specification with the characteristics of the wrapped hand that are responsible. This specification is used to retrieve a standard design object (RIGID-TUBE) to provide those characteristics.

```
<Spec:
Subproblem: Launch
Parts: Spring, Base, RIGID-TUBE
Attached(Spring, Base, <position>)
ENCLOSED(STRING, RIGID-TUBE)
Rigid-Tube:
  SOLIDITY: HOLLOW
  RADIUS: (+ RADIUS(STRING) DELTA)
  LENGTH: (- LENGTH(STRING) SM-DELTA)
  SHAPE: CYLINDRICAL
  LENGTH-VARIABILITY: CONSTANT
  RADIUS-VARIABILITY: CONSTANT
...>
```

Further experimental data from the oracle reveals that the rigid tube hinders compression of the spring, causing the spring displacement limit to be much smaller than expected.

```
ORACLE:
Length-Variability: Constant
Causing
  ; spring displacement is limited by tube length
Actual-x(Spring) <= Rest-Length(Spring)
  - Length(Rigid-Tube)
Causing
  ; ... much less than ideal displacement
Actual-x(Spring) << Max-x(Spring)
```

IMPROVISER detects the undesirable limit on spring displacement and derives the consequences that the launch force will be weaker than is ideal ($F = -kx$), which will affect device speed and striking distance.

IMPROVISER reasons about the causes of the hindrance and updates the specification to require the tube to be collapsible (i.e., allow its length to vary). In other words, IMPROVISER refines the rigidity criterion to what is really needed – the radius to remain constant (lateral-rigidity) and the length to vary (longitudinal flexibility). This requires getting more detailed causal knowledge (not shown) from the oracle about what causes the spring to stay straight, to make sure it won't hurt to vary the length.

```
<Spec:
Subproblem: Launch
Parts: Spring, Base, Rigid-Tube
Rigid-Tube:
  ...
  LENGTH-VARIABILITY: VARIES
  Radius-Variability: Constant
...>
```

Using this specification as a probe to memory, IMPROVISER tries to recall a device that satisfies this specification. The retrieval fails.

Since no viable options are found, IMPROVISER suspends work on the launch subproblem and switches to a different subproblem: how to protect the eggs.

```
<Spec:
Subproblem: Protect-Eggs
Parts: Cushioning-Material
Cushioning-Material:
  Pressure-Resistance: Soft
...>
```

While looking for objects that satisfied this description, a toilet paper holder is observed through an oracle. The observation is a mix of image features and knowledge about the holder, once it has been identified through object recognition.

```
EXTERNAL OBSERVATION: TPH
Structural properties:
  Parts: Cylinders C1 and C2, Spring S
  Associated-Part: Wall-Fixture
  Fits-Inside(C1, C2)
  C1: Solidity: Hollow
    Length: 1/2(Rest-Length(S))...
  C2: Solidity: Hollow
    Length: 1/2(Rest-Length(S))...
  Composition of Cylinders (C1C2)
    Solidity: Hollow
    Length: Length(S) + Delta3
    Shape: Cylindrical ...
  S: Rest-Length > Width(Wall-Fixture)...
  Enclosed(S, C1C2)
Behavioral properties:
  States: Steady, Squeeze, Rest.
  Steady: ; ; bwn sides of wall fixture
    Length(S) < Rest-Length(S)
    x(S) = Length(S) - Rest-Length(S)
    Force-Btwn-Endpts: (- (* k(S) x(S)))
    Length(C1C2) = Width(Wall-Fixture)...
  Squeeze: ; ; being compressed
    Length(C1C2) < Width(Wall-Fixture)
    Length(C1C2) >= Max(Length(C1),
      Length(C2)) ...
  Rest: ; ; outside wall fixture
    Length(C1C2) > Width(Wall-Fixture)
    Length(S) = Rest-Length(S)...
Functional properties:
  Use: Hold paper roll
```

The most relevant problem context is retrieved, indexed by descriptors that are relevant to solving its pending problem (e.g., size and cylindrical shape of tube, spring enclosed in tube). The context contains the descriptors given in the specification for the problem, the options that have been proposed, and the degree to which each matches the descriptors. In this case, the problem context associated with the launch subproblem is retrieved and the paper holder is assimilated into it, based on the structural, imagistic properties of the paper holder and the knowledge associated with it. During this assimilation, IMPROVISER must check whether the tube length can vary by referring to the behavioral knowledge associated with the paper holder to see if the length of the composition of cylinders (C1C2) changes over the states.

A monitor of the assimilation process notices that the paper holder fits the subproblem specification better than any previous option, particularly with respect to variable length.

The results of this process are 1) a problem specification that has been elaborated (with the constraint that the spring must be enclosed in a tube) and refined (with finer-grain constraints on the rigidity properties of the tube), and 2) new knowledge learned about the functions of a common object (i.e., a toilet

paper holder has a new, additional function of keeping its internal spring straight).

Status and Open Issues

Our system currently contains implemented procedures for assimilation of alternatives into a single problem context, evaluation based on specification constraints, and standard memory indexing and retrieval, as well as data structures representing the case library, problem contexts, the evolving problem specification, and the opportunity agenda. Simple monitors surrounding the assimilation process have been implemented, but more are needed for this and the other processes. We currently do not have a general model for what makes some difference or similarity that is noticed "interesting." We are intrigued by the fact that objects can be noticed as being interesting and relevant to a pending problem before their relevance to the problem is fully understood.

As we extend IMPROVISER to handle multiple problem contexts, we need to deal with issues about how to maintain them. For example, how do they decay? What influences which ones are active (e.g., recency, interaction between the problems)? How do they change as related problems are worked on? How does knowledge of functional properties of an object inhibit the retrieval of a relevant problem context in which the object can be used in a new way. (This is important in modeling functional fixedness.)

More work is also needed to identify and define situation assessment procedures, elaboration techniques, and strategic control heuristics. We are also starting to understand how criteria, constraints, preferences, etc., emerge during evaluation, but more effort is needed in modeling this emergence.

Our intention in building IMPROVISER is not to automate design, but to test our hypotheses about the cognition of creative design. As we increase our understanding of creative processes, we will be better able to answer the question how best to assist human designers. This may include 1) aiding the formalization, reformulation, and refinement of specifications (Reubenstein & Waters, 1991; Johnson, Benner, & Harris, 1993), 2) bringing up evaluative issues (Domeshek & Kolodner, 1993), 3) retrieving pending problem contexts to help recognize the applicability of solutions, or 4) proposing new control strategies.

Acknowledgements

We appreciate the insights we have gained from Terry Chandler, Eric Domeshek, Alex Kirlik, Nancy Nersessian, Ashwin Ram, Mimi Recker, and our anonymous reviewers. We would like to thank Otto Baskin, Jon Howard, and Malisa Sarntinoranont, for their invaluable cooperation. This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234.

References

- Bhatta, S., Goel, A., & Prabhakar, S. (1994). Innovation in Analogical Design: A Model-Based Approach. In *3rd Int. Conf. on AI in Design*. Lausanne, Switzerland.
- Domeshek, E.A., & Kolodner, J.L. (1993). Using the points of large cases, *AI EDAM*, 7(2), pp. 87-96.
- Duncker, K. (1945). On problem solving. *Psychological Monographs*, 58(270).
- Grimson, W.E.L. (1990). *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press.
- Hammond, K. (1989). Opportunistic memory. In *IJCAI-89* (pp. 504-510). Detroit, MI.
- Johnson, W.L., Benner, K.M., & Harris, D.R. (1993). Developing Formal Specifications From Informal Requirements. *IEEE Expert*, 8(4), pp. 82-90.
- Jones, E.K. (1992). The Flexible Use of Abstract Knowledge in Planning. (Tech. Rep. 28). Doctoral dissertation. Northwestern University, Institute for the Learning Sciences.
- Jordan, D.S. & Shrager, J. (1991). The role of physical properties in understanding the functionality of objects. In *13th Cognitive Science Conference* (pp. 179-184). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Kolodner, J.L. (1983). Reconstructive Memory: A Computer Model. *Cognitive Science*, 7(4), 281-328.
- Kolodner, J.L. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan-Kaufman Publishers, Inc.
- Kolodner, J.L. & Wills, L.M. (1993) Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design. *AAAI Case-Based Reasoning Workshop* (pp. 19-25).
- Koton, P. (1988). Reasoning about evidence in causal explanation. In *AAAI-88*. MIT Press.
- Lowry, M. (1987). The Abstraction/Implementation Model of Problem Reformulation. In *IJCAI-87* (pp. 1004-1010). Milan, Italy.
- Maier, N.R.F. (1931). Reasoning in humans II: The Solution of a problem and its appearance in consciousness. *Journal of Comparative Psychology*, 12, 181-94.
- Maier, N.R.F. (1970). *Problem Solving and Creativity: In Individuals and Groups*. Belmont, CA: Brooks/Cole Publishing Company, Study 14, pp. 162-175.
- Mueller, E.T. (1990). *Daydreaming in humans and machines*. Norwood, NJ: Ablex Publishing Corporation.
- Navinchandra, D. (1991). *Exploration and Innovation in Design: Towards a Computational Model*. Springer-Verlag.
- Patalano, A., Seifert, C., Hammond, K. (1993). Predictive encoding: Planning for opportunities. In *15th Cognitive Science Conference*, (pp. 800-805). Lawrence Erlbaum
- Reubenstein, H.B. & Waters, R.C. (1991). The Requirements Apprentice: Automated Assistance for Requirements Acquisition. *IEEE Transactions on Software Engineering*, 17(3), pp. 226-240.
- Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press.
- Seifert, C., Meyer, D., Davidson, N., Patalano, A., & Yaniv, I. (1994). Demystification of Cognitive Insight: Opportunistic Assimilation and the Prepared-Mind Perspective. In R.J. Sternberg and J.E. Davidson (eds.), *The Nature of Insight*. Cambridge, MA: MIT Press. Forthcoming.
- Sycara, K. & Navinchandra, D. (1989). Representing and Indexing Design Cases. In *Proceedings of the Second International Conference on Industrial and Engineering Applications of AI and Expert Systems* (pp. 735-741).
- Turner, S.R. (1994). *MINSTREL*. Hillsdale, NJ: Lawrence-Erlbaum Associates, Inc. Forthcoming.
- Wills, L.M. & Kolodner, J.L. (1994) Towards More Creative Case-Based Design Systems. In *AAAI-94*. Seattle, WA.

Towards More Creative Case-Based Design Systems

Linda M. Wills and Janet L. Kolodner

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
linda@cc.gatech.edu, jlk@cc.gatech.edu

Abstract

Case-based reasoning (CBR) has a great deal to offer in supporting creative design, particularly processes that rely heavily on previous design experience, such as framing the problem and evaluating design alternatives. However, most existing CBR systems are not living up to their potential. They tend to adapt and reuse old solutions in routine ways, producing robust but uninspired results. Little research effort has been directed towards the kinds of situation assessment, evaluation, and assimilation processes that facilitate the exploration of ideas and the elaboration and redefinition of problems that are crucial to creative design. Also, their typically rigid control structures do not facilitate the kinds of strategic control and opportunism inherent in creative reasoning. In this paper, we describe the types of behavior we would like case-based design systems to support, based on a study of designers working on a mechanical engineering problem. We show how the standard CBR framework should be extended and we describe an architecture we are developing to experiment with these ideas.¹

Introduction

Creativity in design derives from enumerating several solution alternatives, redescribing and elaborating problem specifications, and evaluating proposed solutions, based on criteria and constraints that go beyond the stated constraints on a solution. It arises out of a confluence of processes (including problem redescription, remembering, assimilation, and evaluation), which interact with each other in complex ways. Often creativity arises from interesting strategic control of these processes, which in themselves may be quite mundane (Boden 1990, Chandrasekaran 1990, Gero & Maher 1993, Navinchandra 1992).

These processes rely heavily on previous design experiences and knowledge of designed artifacts (Goel & Chandrasekaran 1992, Hinrichs 1992, Kolodner & Penberthy 1990, Kolodner & Wills 1993). An expert de-

signer knows of many design experiences, accumulated from personally designing artifacts, being given case studies of designs in school, and observing artifacts designed by others. The designer draws on these experiences to perform such activities as generating design alternatives, reformulating and elaborating the problem specification or proposed solutions, and predicting the outcome of making certain design decisions. The experiences that are most valuable are often those that are highly contextualized pieces of knowledge about artifacts, such as how a device behaves in some context of use, circumstances in which it can fail, and knowledge about situations that might come up not only in use, but in all phases of its life cycle.

Given the nature of these experiences, we believe case-based representations and reasoning techniques lend themselves to supporting creative design. Research in case-based reasoning (CBR) has provided extensive knowledge of how to reuse solutions to old problems in new situations, how to build and search case libraries (for exploration of design alternatives), and how to merge and adapt cases. It has developed powerful techniques for partial matching and the formation of analogical maps between seemingly disparate situations (Kolodner 1993).

However, most existing CBR systems are not living up to their potential. They tend to adapt and reuse old solutions in routine ways, producing robust but uninspired results. They do not attempt to extend their exploration by deriving constraints and preferences that improve or go beyond those stated in the original problem. (See (Kolodner 1993, appendix) for a recent survey.)

Some of this potential is buried in processes that have been downplayed or even missing in most standard CBR systems. In particular, little research effort has been directed towards the kinds of situation assessment, evaluation, and assimilation processes that facilitate the exploration of ideas and the elaboration and redefinition of problems that are crucial to creative design. Also, to facilitate the kinds of opportunism inherent in creative reasoning, CBR systems need to break out of their typically rigid control struc-

¹This research was funded in part by NSF Grant No. IRI-8921256 and ONR Grant No. N00014-92-J-1234.

ture to allow flexible interleaving and communication among processes. In addition, more research attention must be paid to the strategic control mechanisms that guide a creative designer in deciding what to do next.

In this paper, we describe the types of behavior we would like case-based design systems to support, based on an exploratory study of designers working on a mechanical engineering problem. We show how the standard CBR framework should be extended and we describe an architecture we are developing to experiment with these ideas. We end with a set of open issues.

What Do Creative Designers Do?

To gain insights into the knowledge and reasoning involved in creative design, we observed a four-person team engaged in a seven-week undergraduate mechanical engineering (ME) design project. The task was to design and build a device to quickly and safely transport several eggs from one location to another. The device could be constructed from any material, but its size, weight, and cost were restricted.

After exploring several schemes for launching, moving, stopping, and protecting the eggs, the team decided to use a cylindrical egg carrier (of radius 7 cm., length 22.5 cm.), with the eggs wrapped in pipe insulation to protect them inside the carrier. The carrier was dropped down (0.8 m.) from a starting platform and would roll into a target zone (within a 5 m. radius of the starting platform). The team had two possible launch mechanisms up until the final design demonstration day: a spring mechanism and a simple ramp (the spring launch base could be inverted to become a ramp, which was the final choice). In both cases, a string, with one end attached to the launch base, was wrapped around the device, so that as the cylinder dropped, it spun down the string, hit the ground, and rolled into the target zone. The wrapped string gave the carrier momentum and it also prevented it from rolling beyond the target zone.

One of us participated as a member of the team, allowing us to become immersed in the issues and to observe the design process in a natural setting, in both informal and "official" team meetings. We recorded the group's conversations on audiotapes and collected copies of all their design documents and drawings.

We are particularly intrigued by a set of three processes we observed underlying many creative design activities: 1) generation of multiple descriptions or views of a problem, 2) gradual emergence of evaluative issues, constraints and preferences, and 3) serendipitous recognition of solutions to pending problems, sometimes seeing new functions and purposes for common design pieces in the process. We are not claiming that this is a complete set. (For example, our design study has revealed a variety of influences on creativity from collaborative activity.) Rather, we are interested in these processes because they are key processes in design that current case-based systems neglect.

Problem Redescription. The initial problem statement given to our designers was ambiguous, incomplete, contradictory, and underconstrained. They spent a great deal of effort to turn it into something with more detail, more concrete specifications, and more clearly defined and consistent constraints. An important part of this process involved attempting to understand the problem, view it from multiple perspectives, and redescribe it in terms familiar to the designers. They had to refine and operationalize several vague or abstract constraints, while sometimes having to abstract constraints that were too specific.

For example, many of the ideas of one designer, who had a keen interest in automobiles, came from recalling devices and concepts from the car domain, such as shock absorbers, unit-body vs. single-frame construction, and air-bags. Being able to recall these required viewing the problem of protecting the eggs as one of absorbing shock or transferring energy and as a problem of protecting passengers in general, not just eggs.

Our designers also explored the given constraints, deliberately stretching or strengthening them to see what ideas became possible. For example, the initial problem statement was ambiguous about whether or not the device could land (i.e., touch down) short of the target zone and then move into it. The designers considered the extreme possibility of landing as far short of this zone as possible, in which case the device would not fly at all, but would be pushed off or lowered to the ground, where it would then move itself into the safety zone. Visualizing this possibility reminded them of devices, such as elevators and yo-yo's, that could implement parts of this behavior.

This continual elaboration and redescription of the problem helped the designers derive connections between the current problem and similar problems in other domains, facilitating cross-contextual transfer of design ideas. It also primed them to serendipitously recognize relevant objects in the environment that might be reused for a new purpose.

Evaluation. One of the key forces driving evolution of the problem specification is the evaluation of proposed design alternatives. Evaluative issues emerge in the course of evaluating. Designers do not merely depend on constraints that have already been specified. Rather, they bring up additional constraints and criteria as proposals are examined. Proposed solutions often remind them of issues to consider. The problem and solution "co-evolve" (Fischer 1993).

One interesting criteria that emerged in the course of the ME design project was *versatility* – the ability of the device to apply in more than one situation. This criteria was not mentioned or required in the original statement of the problem. It arose in response to ambiguity in the initial problem statement, which described three similar problems but did not specify which one would be assigned. Each problem differed only in the device's starting position (from either the center of a

child's wading pool or from a platform of one of two heights) and in its target destination distance. (This is similar in the real world to situations in which the engineers are designing for multiple potential customers with different needs). To deal with the uncertainty and reduce the complexity this variability introduced, the designers began searching for solutions that could be used to solve all three problems or could be easily adapted to apply to each. That is, they began to evaluate proposals on the basis of versatility in addition to the other criteria already in the problem specification. Being able to do this is central to creative design.

Assimilation. Problem redescription provides not only a means for recalling relevant solution alternatives, but also a vocabulary for describing and, in many cases, reinterpreting objects in the designer's environment. This often leads to a new way of viewing the function of some object and facilitates the recognition of potential solutions to pending problems in the external environment.

For example, our designers went to a home improvement store for materials for a spring launch mechanism. While comparing the strengths of several springs by compressing them, they noticed that the springs tended to bend. One designer wrapped a hand around the spring to hold it straight as it was compressed and said the springs would each need to be enclosed in a tube to keep them from bending. Another added that the tube would need to be collapsible (to compress with the spring). The designers could not think of an existing collapsible tube and did not want to build one due to time pressure. They gave up on the springs and started thinking about egg protection. During their search for protection material, they walked through the bathroom section of the store, where they saw a display of toilet paper holders. They immediately recognized them as collapsible tubes which could be used to support the springs.

By playing with the springs, noticing problems and suggesting fixes, the designers formed a specific, concrete, and operationalized description of what a solution would look like to the bending-springs problem. However, the toilet-paper holder was not recalled on the basis of this description. Instead, the description was used to reinterpret the toilet paper holder when it was encountered in the external environment and to recognize its additional function of preventing springs from bending upon compression. The designers were able to interpret objects seen in the environment, or recalled from memory, from a new viewpoint. This viewpoint was based on descriptions and feature dimensions that had been revealed to be important in attempts to solve recent and pending problems.

We refer to this process as *assimilating* the objects into a problem context. It not only involves reinterpreting solution alternatives under consideration, but also comparing and contrasting alternatives with one another, along the dimensions relevant to the problem

context. This helps reveal those that are not really new ideas, so that they can be ignored. It can also cause new evaluative issues to emerge as new dimensions or criteria are generated to distinguish seemingly identical ideas.

Strategic Control. The designers we observed did not follow a rigid, methodical plan detailing what to do next. Rather, they moved fluidly between various problem pieces and design processes (e.g., idea generation, adaptation, critiquing, problem refinement, elaboration, and redefinition) in a flexible and highly opportunistic manner.

Our designers employed a variety of strategic control heuristics, some of which are opportunistic. For example, when an alternative was proposed that satisfied some desired criteria extremely well compared to the other alternatives, they directed their efforts toward elaborating that alternative, optimistically suspending criticism or discounting the importance of criteria or constraints that were not satisfied as well. Sometimes this led to reformulation of the problem as constraints were relaxed or placed at a lower priority.

Being able to take advantage of such opportunities requires being able to judge whether progress was being made along a certain line of attack and to choose which ideas are more promising or more likely to lead to something unusual and novel.

Some strategic control heuristics are more deliberate, based on reflection. For example, one heuristic our designers used was to try quick, easy adaptations of a proposed solution first before stepping back and reformulating the problem or relaxing constraints. Other deliberate heuristics attempted to make non-standard substitutions, apply adaptation strategies in circumstances other than the ones they were meant for, and merge pieces of separate solutions with each other in nonobvious ways.

In many cases, the processes that are composed together leading to a novel idea are not in themselves novel and may be quite mundane. The trick is knowing when to do them.

How CBR Systems Can Do Better

Most current CBR systems tend to stick to well-known interpretations of problems and routine ways of adapting old solutions, neglecting exploration of alternatives if something good enough has been found. We believe the CBR paradigm can be extended to support more creative problem solving.

Problem Redescription. Problem redescription corresponds closely to the process of situation assessment – redescrining a problem in the vocabulary of the indexing system. In most CBR systems, situation assessment is skipped; the assumption is made that the initial representation of the problem is sufficient for solving the problem. But, as our observations show, investigating a problem in depth makes available a large

set of relevant cues for retrieval. Generating multiple ways of describing a problem provides several different contexts for specifying what would be relevant, if remembered.

Research on indexing has found that it is the combination of setting up a context for retrieval and having already interpreted something in memory in a similar way that allows retrieval. When some case or piece of knowledge is entered into memory, it is not always possible to anticipate how it might be used. Situation assessment processes aim to bridge that gap by helping to redescribe a new problem in a way that is similar to something seen before.

Research into situation assessment and problem reformulation (e.g., in CASEY (Koton 1988), CYRUS (Kolodner 1983), MINSTREL (Turner 1994), BRAINSTORMER (Jones 1992), and STRATA (Lowry 1987)), show different ways it can be done. However, these techniques have not yet made it into widespread use in practical CBR systems. They should certainly be included in any system aimed at reuse of experience across domains.

Evaluation. CBR systems currently evaluate solutions by checking a set of constraints that have been given to the system. Evaluative procedures are typically buried within case manipulation to predict or test whether a modified case satisfies the specified constraints. Observations of our designers suggests that evaluation should play a more prominent role in case-based design systems, allowing evaluative issues to emerge in the course of evaluating. Navinchandra (1991) calls this *criteria emergence* and shows an example of how it can arise from case-based projection. In addition to criteria, constraints in general (Prabhakar & Goel 1992) and relative priorities among them also gradually emerge. This type of evaluation is a key driving force within creative design, feeding back to situation assessment and guiding case manipulation.

Assimilation. A key idea underlying dynamic memory (Schank 1982), one of the principle foundations of case-based reasoning, is that remembering, understanding, and learning are all inextricably intertwined. The ability to determine where something fits in with what we already know (understanding) is a key part of being able to assimilate objects in our environment into our problem solving. This environment includes not only external objects, but also cases that have been retrieved, elaborated and adapted. Understanding how these fit into a problem context may involve a useful reinterpretation of something already in memory, suggesting in a new way of indexing it.

Strategic Control. Our exploratory study suggests that a linear, sequential composition of CBR processes is much too simple. In reality, these processes are highly intertwined and interact in interesting ways. For example, problem elaboration and redescription tac-

tics specify contexts for search that retrieval processes use, while evaluation of recalled or adapted alternatives feeds information back to these situation assessment tactics, resulting in even better contexts for search. In some cases, what suggests a particular problem refinement or redescription results from trying to confirm the legality of a proposed solution during evaluation and finding a loophole or ambiguity in the current problem specification. In addition, comparing and contrasting a proposed solution with other proposals during assimilation can bring new evaluative issues into focus.

CBR systems need to break out of their typically rigid control structure and allow more interaction and opportunism among processes. This requires making strategic control mechanisms explicit, so they can be easily modified, reasoned about, extended, and learned. More research needs to be directed at identifying and capturing the types of strategic control heuristics designers use.

Proposed Architecture

We are developing an experimental case-based system that emphasizes the processes of situation assessment, evaluation, and assimilation, integrating them with the usual CBR processes of retrieval, elaboration (case manipulation, adaptation, merging, prediction), and learning. It has a flexible, opportunistic control structure which allows us to keep control tactics separate, explicit, and modifiable.

The processes within our system are not applied in a strictly linear succession. Rather, the system has a blackboard-style architecture. The processes are centered around and act upon data structures that represent the evolving problem specification and the set of design alternatives under consideration.

Situation assessment procedures act on the problem specification to evolve it along multiple directions. Evaluation examines design alternatives, checking them against the current specification, to reveal inconsistencies, ambiguities, and incompletenesses in the specification that suggest new redescription. Evaluation also brings up new criteria, and constraints which are incorporated into the problem specification.

Elaboration procedures transform alternatives under consideration into new alternatives by applying a variety of adaptation and merging strategies. These strategies are typically suggested by the critique formed by an evaluation of some alternative. Elaboration procedures also augment alternatives with information derived about their consequences and expected behavior. These "data collection" elaborations are currently accomplished by manual augmentations of alternatives with experimental data, but in general can be achieved by case-based projection, simulation, actual experimentation, or visualization.

The evolving problem description is also used by both the retrieval and the assimilation processes. Retrieval interfaces with a library of cases which models,

in part, long-term memory. The problem description is used as a probe into memory to pull relevant design cases into consideration (for evaluation, elaboration, etc.). The assimilation process is the dual of retrieval. It accumulates design alternatives proposed (i.e., those retrieved, elaborated, or viewed directly in the external environment) into the pool of design alternatives under consideration, organizing the alternatives with respect to each other.

The data structure holding the set of design alternatives forms an extension of the long-term memory. We call this extension the "problem context." The evolving problem description determines the focal vocabulary of the current problem context. As the specification evolves, the focus changes on the relevant vocabulary to be used for organizing alternatives in the memory (e.g., shape, construction cost, personal safety). In a sense, the problem context is providing a point of view with respect to which objects in the environment and cases recalled can be interpreted and organized by the assimilation process.

The coordination of the various processes is controlled by explicit strategic control mechanisms. There are a set of monitoring procedures, associated with each of the processes, which watch for opportunities for some task to be performed. The opportunities noticed are placed on an "opportunity agenda." Opportunities are chosen and pulled from the agenda by strategic control heuristics. For example, a monitor associated with the assimilation process watches for an alternative to be added that is much better than any other alternative proposed so far, with respect to some desired criterion. This yields an opportunity to change the problem description by increasing the priority of that criterion and/or by relaxing constraints that are not met by that proposal. This simulates the behavior of changing the relative importance among criteria to accommodate an unexpectedly good solution that is stumbled upon. An example strategic control heuristic would be to pursue elaboration opportunities for alternatives that satisfy a desired criteria extremely well before pursuing evaluative processes that would negatively critique the alternatives. This simulates the behavior of optimistically pursuing an idea, suspending all but constructive criticism.

Status, Limitations and Open Issues

Our system currently has implemented procedures for evaluation, assimilation, and retrieval, as well as data structures representing the case library, pool of design alternatives, evolving problem specification, and the opportunity agenda data structure. We have standard agenda management routines. However, these routines currently do not model the ephemeral nature of opportunities (which can either expire or be forgotten). Several monitors surrounding the assimilation process have been implemented, but we still need to define and capture those relevant to the other processes.

Much more work is needed to identify and define strategic control heuristics, situation assessment procedures, and elaboration techniques. Also, not all strategic control mechanisms are triggered by noticing an opportunity. Some may become applicable due to some complex condition that must be inferred through reflection. (For example, realizing that you are reasoning in circles might cause you to make an effort to try a brand new technique.) More research needs to focus on how to represent and infer these kinds of conditions and also how the application of these more reflective strategic control mechanisms can be interleaved with the triggering of opportunistic ones.

We are starting to understand how criteria, constraints, preferences, etc., emerge during evaluation, but more effort is needed in modeling this emergence.

There are a number of interesting open issues concerning how assimilation is managed when the design problem is complex, having several interacting subproblems, each of which have different sets of alternatives and requirements. Assimilation must find the appropriate problem context for interpreting and evaluating a given design alternative. The ability to do this facilitates the serendipitous recognition of solutions to pending problems, as we saw in the bending-springs problem. (See also (Seifert et al. 1994).)

Another open issue is that the designers we studied were not expert mechanical engineers. An interesting empirical question is: would experts, having knowledge of "design principles," behave differently? It may not be the expert vs. novice distinction, but how open-ended the problem is, that is important. After all, the students were familiar with and experienced in solving everyday mechanical problems using objects in their world. We believe that for open-ended, nonroutine problems, expert designers are likely to display the same sorts of behaviors as do our students.

Finally, there are some aspects of creative design that we have not yet explored. In particular, we would like to analyze more carefully the influences collaboration had on creativity in the design project. Our agenda-based model of opportunity management lends itself to simulating the exploration of several opportunities in parallel, and employing multiple control strategies at once. This will allow us to simulate these aspects of collaborative activity and use computational experiments to explore hypotheses about the role of collaboration in creative design.

Conclusion

Our intention in building our system is not to automate design, but to test our hypotheses about the cognition of creative design. We are trying to understand creative processes better, using a case-based cognitive model. As we increase our understanding (and in the process, push CBR technology), we will be able to answer the question how best to assist human designers. This may include 1) aiding the formalization, reformu-

lation, and refinement of specifications (Reubenstein & Waters 1991, Johnson, Benner, & Harris 1993), 2) bringing up evaluative issues (Domeshek & Kolodner 1993), 3) retrieving pending problem contexts to help recognize the applicability of solutions, or 4) proposing new control strategies.

We are taking a case-based approach to understanding creative design for two reasons. One is that many creative design activities are highly memory-intensive and rely on past design experiences, so case-based reasoning has much to offer in this study. The other is that we hope to make case-based systems themselves more creative. By using the paradigmatic tools CBR provides, we are starting to find computational models of the behaviors and processes we observed in our exploratory study. At the same time, our modeling attempts have deepened our understanding of case-based processes and memory issues and have suggested extensions that will yield more creative design systems in the future.

Acknowledgements

We appreciate the insightful discussions we have had with Terry Chandler, Eric Domeshek, Lucy Gibson, Todd Griffith, Kenneth Moorman, Nancy Nersessian, Ashwin Ram, and Mimi Recker. We would like to thank Otto Baskin, Jon Howard, and Malisa Sarntinoranont, for their invaluable cooperation. We also appreciate the insights and helpful comments of our anonymous reviewers.

References

- Boden, M. 1990. *The Creative Mind: Myths and Mechanisms*. New York, NY: Basic Books.
- Chandrasekaran, B. 1990. Design Problem Solving: A Task Analysis. *AI Magazine*. 11(4): 59-71.
- Domeshek, E.A., and Kolodner, J.L. 1993. Using the points of large cases, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 7(2): 87-96.
- Fischer, G. 1993. Turning Breakdowns into Opportunities for Creativity. In Proceedings of the International Symposium on Creativity and Cognition, Loughborough, England.
- Gero, J. and Maher, M. 1993. *Modeling Creativity and Knowledge-Based Creative Design*. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.
- Goel, A. and Chandrasekaran, B. 1992. Case-based Design: A Task Analysis. In C. Tong and D. Sriram (eds.), *Artificial Intelligence Approaches to Engineering Design, Volume 2: Innovative Design*. San Diego, CA: Academic Press.
- Hinrichs, T. 1992. *Problem Solving in Open Worlds: A Case Study in Design*. Northvale, NJ: Erlbaum.
- Johnson, W.L., Benner, K.M., and Harris, D.R. 1993. Developing Formal Specifications From Informal Requirements. *IEEE Expert* 8(4): 82-90.
- Jones, E.K. 1992. The Flexible Use of Abstract Knowledge in Planning. Northwestern University, Institute for the Learning Sciences Technical Report no. 28.
- Kolodner, J.L. 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan-Kaufman Publishers, Inc.
- Kolodner, J. 1983. Reconstructive Memory: A Computer Model. *Cognitive Science* 7(4): 281-328.
- Kolodner, J.L. and Penberthy, T.L. 1990. A Case-Based Approach to Creativity in Problem Solving. In Proceedings of the Twelfth Annual Conference of the Cognitive Science Society, Cambridge, MA.
- Kolodner, J.L. and Wills, L.M. 1993. Case-Based Creative Design. In AAAI Spring Symposium on AI and Creativity. Stanford, CA. Reprinted in *AISB Quarterly* 85: 50-57.
- Koton, P. 1988. Reasoning about evidence in causal explanation. In Proceedings of the 6th National Conference on Artificial Intelligence. Cambridge, MA: AAAI Press/MIT Press.
- Lowry, M. 1987. The Abstraction/Implementation Model of Problem Reformulation. In Proceedings of the 10th International Joint Conference on Artificial Intelligence, pp. 1004-1010. Milan, Italy.
- Navinchandra, D. 1991. *Exploration and Innovation in Design: Towards a Computational Model*. New York: Springer-Verlag.
- Navinchandra, D. 1992. Innovative Design Systems: Where are we, and where do we go from here?. Parts I and II. *The Knowledge Engineering Review* 7(3): 183-213 and 7(4): 345-362.
- Prabhakar, S. and Goel, A. 1992. Performance-Driven Creativity in Design: Constraint Discovery, Model Revision, and Case Composition. In Proceedings of the Second International Conference on Computational Models of Creative Design. Heron Island, Australia.
- Reubenstein, H.B. and Waters, R.C. 1991. The Requirements Apprentice: Automated Assistance for Requirements Acquisition. *IEEE Transactions on Software Engineering* 17(3): 226-240.
- Schank, R. 1982. *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press.
- Seifert, C., Meyer, D., Davidson, N., Patalano, A., and Yaniv, I. 1994. Demystification of Cognitive Insight: Opportunistic Assimilation and the Prepared-Mind Perspective. In R.J. Sternberg and J.E. Davidson (eds.), *The Nature of Insight*. Cambridge, MA: MIT Press. Forthcoming.
- Turner, S.R. 1994. *MINSTREL*, Lawrence-Erlbaum Associates, Inc. Forthcoming.

Understanding the Creative Mind

Ashwin Ram
Linda Wills
Eric Domeshek
Nancy Nersessian
Janet Kolodner
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

We review Margaret Boden's book *The Creative Mind*, an excellent survey and synthesis of current computational theories of creativity. Boden's stated goal is to explain how creativity (as a psychological phenomenon) is possible, where an explanation of possibility is taken to be a computational process. Although Boden does not deliver a full-fledged computational explanation and leaves most details of the underlying processes unexplicated, she provides a strong argument that such an explanation is possible.

As part of our critique, we sketch our preferred (case-based) framework for modeling creativity, in which much of mental life depends on the retrieval and manipulation of past experiences. We focus on five major influences on cognition (and thus on creativity): inference, knowledge, task, situation, and strategic control. We also highlight "constructive modeling" which integrates analogical reasoning with visual reasoning and thought experimentation.

Our framework, while broadly compatible with Boden's, is more specific in its suggestions for integrating multiple types of interacting and interactive processes. We emphasize issues of control and the role of experience. By focusing on how mental activity is directed towards a task in some situation, we ensure that the resulting theory addresses pragmatic issues in thinking and control of thinking.

To appear in *Artificial Intelligence* journal. In addition, a shorter review of Boden's book by the same authors will appear in the journal *Behavioral and Brain Sciences*.

1 Computational Creativity

Margaret Boden, a master at bringing ideas from artificial intelligence and cognitive science to the masses, has done it again. In *The Creative Mind* (Boden, 1990), she has produced a well-written, well-argued review and synthesis of current computational theories relevant to creativity. This book seems appropriately pitched for students in survey courses and for the intelligent lay public. And if ever there were a topic suitable for bridging the gap between the researchers and the layperson, this is surely it: What is creativity, and how is it possible? Or, in computational terms (the terms that Boden argues ought to be applied): what are the processes of creativity?

Boden's stated goal is to explain how creativity is possible, where creativity is taken to be a psychological phenomenon, and an explanation of possibility is taken to be a computational process. As computationalists with active interests in creativity, we find this perspective congenial. But while offering many examples of creativity and surveying many approaches to creativity, the book leaves most details of the processes of creativity and their interactions unexplicated. Nevertheless, although Boden does not deliver a full-fledged computational explanation of the phenomenon, she does provide a strong argument that such an explanation is possible.

The early motivational sections of the book enthusiastically play up the notion that creativity, as opposed to "mere novelty," is somehow paradoxical. The middle section offers broad but shallow coverage of existing computational models with some emphasis on connectionist approaches. The exposition is studded with excellent examples of creativity drawn from the worlds of high culture and epochal science. Major chapters are devoted to "Unromantic Artists" and "Computer Scientists" – that is, to computer programs that have been built to simulate artistic and scientific creation and interpretation (e.g. AARON, BORIS, TAIL-SPIN, ARCS, DENDRAL, BACON, AM, and so on). The final section dwells on a grab-bag of ancillary issues such as the relationship of randomness to creativity, the degree to which creativity is reasonably conceived as a special gift limited to the chosen few, and the nature of computational theories and explanations of phenomena such as creativity (including a round in Searle's Chinese Room).

Our interest is primarily in the examples, implementations, and theories that comprise the middle section of the book, but it is important to spend a bit of time understanding the problem as Boden has laid it out. As noted, Boden's goal is a computational account of the psychological phenomenon of creativity. Her achievement is to make the possibility of such a theory seem more probable (or, perhaps, at least conceivable). Both goal and achievement, however, must be contrasted with other possible ends. Boden designates the object of her study as "P-creativity" (or *psychological creativity*), distinguishing it from at least two other related concepts: "mere novelty" and "H-creativity" (or *historical creativity*).

P-creativity is a cognitive notion. By asking how some individual came up with an idea that seems beyond what they ought to be able to think, one concerns oneself with thought processes, and can deploy all the tools of computational modeling to understand these processes. In contrast, H-creativity refers to judgments that are made by a culture about the novelty and worth of ideas. Boden downplays the value of this standard, arguing convincingly that H-creativity is overly restrictive, and that P-creativity is the more significant in that H-creativity typically results from it. Boden chooses many H-creative ideas as

glamorous examples, but the assumption remains that most instances of H-creativity must in the end be explained in terms of some individual's P-creative act. We agree that the important scientific question is how P-creativity could happen and that the right kind of answer to this question is a computational one. After all, no one has much of a handle on a computational model of culture. The key distinction between P-creativity and H-creativity is Boden's position that creativity is an attribute of mental *processes* rather than mental *products*. Although there is consensus that historically significant innovations are creative, Boden holds that what is creative when thought by one individual may not be so when thought by another. As computationalists, we like this emphasis on process over product in defining creativity.

Furthermore, we believe that a creative outcome is not the outcome of extraordinary mental processes, but of mechanisms that are on a continuum with those used in ordinary thinking. In our view (and Boden's), extraordinary outcomes arise from the application of ordinary mechanisms, enhanced and applied with conscious (strategic) control. For example, later in this review we describe Maxwell's use of analogy in deriving the electromagnetic field equations. In doing so, Maxwell constructed a hybrid analogical source model for electromagnetism that draws physical and mathematical constraints from two mechanical source domains: continuum mechanics and machine mechanics. It is not ordinary to construct a hybrid hypothetical analogy as Maxwell did, but analogy is an ordinary mechanism. To understand creativity, we need to understand what is different about the employment of ordinary mechanisms in creative problem solving. The focus on the outcome, for example, as in historical creativity, may provide criteria for what counts as a creative idea, but not an understanding of what is a creative reasoning process. (We will return to this point later.)

Boden's distinction between psychological and historical creativity is important (in fact, indispensable) in establishing the book's focus. Opposing P-creativity to mere novelty is also important. It serves to rule out easy, boring cases of new ideas that are not interestingly new. How Boden makes this distinction, however, strikes us as somewhat problematic. Boden argues that true creativity (as opposed to mere novelty) occurs when a person thinks a thought that is outside the space of thoughts that are even conceivable to that person – outside, as it were, their knowledge level (Newell, 1982). To clarify this idea, she invokes representations, rules, and search spaces, noting that fixing these constructs limits what can be generated by the thought processes of the reasoner. Creativity, then, requires the modification of these structures in order to expand their generative capacity.

Notice, however, that these clarifications have the effect of building aspects of a particular computational account of mental life into the definition of creativity. The effect is to limit the range of computational explanations up for consideration to those that are expressible within the particular computational paradigm chosen to model the mind. We believe the choice of constructs playing a role in mentation (and thus up for modification) are subject to debate; as will be elaborated below, we would invoke constructs such as cases, indexing structures, adaptation rules, and control strategies.

The balance of this review is organized as follows. In the next section we offer an initial critique of Boden's approach to characterizing creativity, and raise a set of questions we believe must ultimately be addressed (though we certainly do not claim to be able to answer all of them). In Section 3. we lay out our preferred framework for thinking about and modeling creativity – a framework in which much of mental life depends on the retrieval

and manipulation of past experiences. Within this *case-based reasoning* framework, we focus on five major influences on cognition (and thus on the potential for creativity); each of these five influences is illustrated using examples of mechanical design, the first of three different domains we have studied, and is related to some of Boden's observations. Section 4 takes up some of the issues raised in our early critique of Boden's model, using examples of everyday creative interpretation (our second domain) to argue against the notion of special creative processes. Section 5 focuses on "constructive modeling," which integrates analogical reasoning with visual reasoning and thought experimentation. The value of this process and how it fits into our framework is illustrated by an example of historically (and psychologically) important scientific creativity (our third and final domain). Section 6 concludes this review by summarizing our approach to modeling creativity and relating it to Boden's position.

2 Characterizing the "Thinkable"

Although we disagree with Boden's choice of constructs, one needs *some* characterization of the space of thoughts that are *ordinarily thinkable* by the computational model, and the set of modifications to the thought-generating elements in the model that modify this space in an interesting manner. Ideally, what counts as an interesting modification should be specified in a manner independent of the particular computational modeling paradigm, although the modification mechanisms themselves can, of course, only be specified in the chosen formalism. In other words, the issue is: whatever the constructs involved in mentation, be they cases, rules, or search spaces, what counts as the "ordinarily thinkable," and what counts as a "creative" (as opposed to mundane) modification of the space of ordinarily thinkable thoughts?

We agree with Boden in that she refrains from defining the thinkable in terms of what is derivable through deduction from the reasoner's knowledge (as, for example, is often done in formalizations of "knowledge levels" (Dietterich, 1986; Newell, 1982)). Instead, the search space includes everything derivable from all the available reasoning operators (which could, and usually do, go beyond deduction). However, this leads to the paradox that, in some sense, every thought must be part of the set of thoughts that could be generated through available reasoning operators; if one comes to think a thought, it must have been thinkable. Boden's answer to this is that some operators carry out *conceptual change* (Carey, 1985; Nersessian, 1992; Ram, 1993; Thagard, 1992) and thus fundamentally modify the search space.

This account falls short in two ways: first, conceptual change is as elusive a notion as creativity itself (Nersessian, 1992), and second, it is not obvious why the search space generated by application of conceptual change operators is not considered part of the thinkable. An independent (and operationalized) characterization of what makes these conceptual change operators different from all the other more ordinary inferential operations is needed.

In particular, consider Boden's formulation of thought as a search over a given search space defined by a set of constraints, operators, and representations. Boden implies that creative search involves changing or extending the constraints, operators, or representation, using an additional set of operators (with associated constraints and representations) whose job it is to modify the first set. Thus, ordinary thought is a search over an ordinary (albeit

non-deductive) search space, whereas creative thought is a *meta-search* using a separate set of operators. While such an account, in principle, is perfectly acceptable, it is unclear what theoretical principles would license the placement of a given operator (or piece of knowledge) into one or the other of the search or meta-search categories. As we will elaborate below, we do not believe there are special meta-search operators that are different from ordinary inferential mechanisms.

Furthermore, we are skeptical that those individuals noted for producing many interesting ideas undergo radical conceptual change in order to produce each idea. Although this may be true of many historically significant ideas, we would prefer a model of *long-term conceptual development* in which the individual evolves a search space, that, when explored by *normal* thought processes, still includes many thoughts that would be considered *creative*.

These objections notwithstanding, we are fully sympathetic with Boden's goal of explaining creativity by appeal to computational processes. We were, therefore, most interested in the particular set of processes suggested: heuristic search (as in BACON), multiple levels of representation (as in BORIS), fuzzy matching (using an unspecified connectionist implementation), and most notable, conceptual change (unimplemented).

We agree with the idea of creativity emerging through multiple interacting processes, but we think that Boden's account leaves open several questions. First, the discussion of the mechanisms, though suggestive, is more descriptive than computational. Second, it is unclear what the overall process model is: How do all these mechanisms fit together? How do they interact? Do they operate on the same representations? If not, how do they communicate, and what do they communicate about? A third set of issues relates to Boden's suggestion that these processes are not unique to specially endowed individuals. It is never quite clear whether these processes are unique to creative thought, or, if not, what distinguishes those thoughts that are creative from those that are not, within a single individual.

3 Five Aspects of Thought

Parallel to and independent of Boden's analysis, we have been studying creative reasoning in several different domains, with a similar goal of producing computational process models of creativity. Much of what we have found concurs with Boden's observations and proposals, but we are seeking more specifics and more coherence in our models. We believe that in order to analyze creative reasoning, one needs a theoretical computational framework in which to model thinking. To this end, we propose using a computational approach rooted in case-based reasoning (Kolodner, 1993). This paradigm is fundamentally concerned with memory issues, such as reminders from partial matches at varying levels of representation and the formation of analogical maps between seemingly disparate situations – exactly the kinds of phenomena that researchers up to, and including, Boden have highlighted as central to creativity.

Accordingly, we see creative thought, like all thought, as involving processes of problem interpretation and problem reformulation, case and model retrieval, elaboration and adaptation, and ultimately, evaluation. Interpretation and reformulation are part of *situation assessment* – the process of redescribing a problem in the vocabulary of a memory's indexing scheme. Elaboration and adaptation include standard analogical processes as well as the

more general process of constructive modeling, discussed at length in Section 5. Evaluation includes outcome determination, be it by simulation or by case-based prediction. All of these processes follow from our enriched case-based reasoning model (Kolodner, 1994), and fit together into a coherent whole within that framework. Research in case-based reasoning has provided extensive knowledge of how to analyze and reformulate problems, how to reuse solutions to old problems in new situations, how to build and search libraries of experience, how to merge and adapt experiences, and how to evaluate candidate solutions.

Our examples of creativity are drawn from three disparate domains: We are studying creativity in the everyday activities of average people by studying the design of mechanical devices (Kolodner & Wills, 1993a; Wills & Kolodner, 1994a, 1994b) and by looking at the processes involved in reading and understanding science fiction stories (Moorman & Ram, 1994a, 1994b; Ram, 1993). At the same time, we are examining and analyzing what led to the significant scientific discoveries of Maxwell and Faraday (Nersessian, 1984, 1992, 1993). Examples drawn from these studies, as well as Boden's own examples, will be used to illustrate our points.

Our research suggests that creativity is not a process in itself that can be turned on or off; rather, it arises from the confluence and complex interaction of *inferences* using multiple kinds of *knowledge* in the context of a *task* or problem and in the context of a specific *situation*. Much of what we think of as creativity arises from interesting *strategic control* of these inferences and their integration in the context of a task and situation. These five aspects – inferences, knowledge, task, situation and control – are not special or unique to creativity but are part of normal everyday thinking. They determine the *thinkable*, the thoughts that the reasoner might normally have when addressing a problem or performing a task.

To give a taste of what we mean by each of these five aspects, the next five sections give examples of each aspect in the context of design. Design is a pervasive form of thinking which most people do every day, not just in specific engineering contexts. All five aspects of thought are involved in design reasoning along the entire continuum from routine to creative design. The goal of this section is to give examples of the five aspects that determine the thinkable. The next section discusses what it means to go beyond the thinkable with respect to these five aspects.

3.1 Inferential Mechanisms

We have performed an exploratory study in which we observed a four-person team engaged in a seven-week undergraduate mechanical engineering (ME) design project (Kolodner & Wills, 1993a; Wills & Kolodner, 1994a). The task was to design and build a device to quickly and safely transport several eggs from one location to another. In this study, we observed that designers move fluidly between a variety of inferential methods. Typical ones include problem understanding, decomposition, elaboration, and redescription, as well as remembering, adapting, and merging design artifacts previously seen.

For example, while trying to think of ways of launching a heavy transport device, carrying several eggs from a pool of water, our ME designers recalled the behavior of a submarine submerging and launching a missile. This helped them to visualize the desired behavior of the device being designed and to elaborate the problem specification. While visualizing and

acting out the missile launch, the students noticed that submarines launch missiles one at a time. This led to a redescription of the problem from launching a group of eggs in a single launch to launching each egg individually in multiple launches. The students went on to merge this idea with other ideas they had earlier, such as enclosing each egg in a tennis ball for protection (an adaptation of an earlier idea to enclose several eggs in a NERF football).

Such inferences are driven and guided by the evaluation of proposed design ideas through critical analysis, as well as by experimentation and mental simulation. The generative mechanisms, guided by critiques, respond to opportunities to create new alternatives by merging or adapting proposed ideas. The design specification is incrementally updated as ideas are tested and flaws or desirable features become apparent.

The types of inferential methods we observed (e.g., problem elaboration and redescription, solution remembering, adapting, and merging) were applied throughout the design process to produce routine (thinkable) as well as innovative ideas. They were applied in a flexible and highly opportunistic manner, with their application heavily influenced by the other four aspects of thought. Computational models of several inferential mechanisms exist, which exemplify the inferential aspect of thought. These include:

- reinterpretation of an idea in terms of a different but familiar idea (e.g., Jones (1992) shows how this can lead to useful problem reformulations which facilitate the operationalization of abstract advice (in the form of proverbs) during planning situations),
- visualization, mental simulation, and thought experimentation, which we have seen to be useful in evaluating and elaborating ideas, and in reformulating problems in design (Kolodner & Wills, 1993b) and scientific reasoning (Nersessian, 1992, 1993),
- constraint relaxation and substitution, which is useful in problem reformulation and elaboration (e.g., Moorman & Ram (1994a) show how new concepts can be formed or understood, while reading science fiction stories, by systematically tweaking constraints on known, familiar objects),
- relaxing constraints during memory search, which facilitates problem reformulation and retrieval (e.g., Turner (1994) calls this *imaginative retrieval* and shows how it can be used to retrieve ideas for writing short stories),
- relevance assessment, which is useful, for example, in retrieval and evaluation (Ram & Leake, 1991), and
- explanation of anomalies, which is also useful in retrieval and evaluation (e.g., (Ram, 1994; Schank, 1986)).

3.2 Knowledge Sources

Our second aspect of thought is knowledge. Designers draw on a variety of knowledge sources, particularly previous design experiences, accumulated from personally designing artifacts, studying case studies of designs in school, and observing artifacts designed by others. Designers typically work within a "design culture" (Navinchandra, 1992) of common engineering practices, design styles, techniques, and technologies. Innovation often arises when ideas from one culture are applied in another. In our ME design study, one designer

drew much inspiration from automotive engineering, a design culture in which he is intensely interested. Many of his ideas came from recalling devices and concepts from the car domain, such as shock absorbers, unit-body versus single-frame construction, and air-bags.

A crucial part of what makes this transfer possible involves understanding, elaborating, and redefining the given problem specification to make connections to domains with which they are familiar. Designers often build on their knowledge of previous, similar problems (and their solutions) to derive new constraints and priority structures that improve or go beyond those stated in the original problem description. For example, our ME designers redefined their launch problem, based on recalling how submarines launch missiles. They derived evaluative issues and new criteria and constraints, based on their experiences with devices such as cars, toys and sports equipment, as well as designs for previous high-school egg-drop projects.

Many of the aspects of constraint exploration we observed in our designers can be experienced by Boden's reader when, in Chapter 4, she encourages the reader to play a game of necklace building within a set of rules. As Boden points out, the construction and exploration of conceptual spaces is often facilitated by drawing analogies to familiar concepts so that knowledge and reasoning techniques can be transferred to the current problem. As we will show later, the same sorts of redescription and construction of conceptual structures occur in the other two areas we have studied – science fiction reading (in which new concepts must be invented to understand the stories) and scientific discovery (in which new hybrid models are designed by merging pieces of knowledge from multiple source domains). We call this process *constructive modeling* (Clement, 1989; Moorman & Ram, 1994b; Nersessian, 1992, 1993, in press; Nersessian & Greeno, in process). Other existing mechanisms for accessing and manipulating knowledge sources include redescription and abstraction, such as reinterpretation of data at a higher level (for example, symbolic interpretation of numerical data (Ram, 1993; Kuipers & Byun, 1991)), and cross-contextual analogy (e.g., (Ram, 1993; Schank, 1982)).

Transferring knowledge from one design culture (or domain, in general) to another is not necessarily P-creative. However, identifying a domain as relevant, figuring out which pieces of knowledge or which strategies can be transferred to a new problem, and how to adapt and combine them to solve the new problem can be a creative process. These are important questions of focus which Boden does not address, but which are central to understanding what guides exploration within a generative system. (Boden is concerned more with how creativity is possible than with what guidance can make it more probable.) We believe many of the answers to these focus-related questions come from the task at hand and the situational context.

3.3 Tasks

A third aspect influencing what is thinkable is the task. Design is a complex task, involving several subtasks, such as brainstorming, critiquing, gathering information about and elaborating ideas, and finding, constructing, and integrating design pieces. Which aspects of a remembered design experience or a proposed design alternative the designer focuses on depend on what is relevant to the task at hand. This can greatly influence the strategic control of the design process, as well as which new constraints or criteria are added to the

design specification and which elaborations or adaptations of ideas are suggested.

For example, there are numerous facts associated with submarines, but our designers were drawn to the fact that they launch missiles one at a time, as opposed to, for example, facts about how missiles are aimed at their target or about the cramped, claustrophobic interior. They were viewing the submarine missile launch from the perspective of trying to borrow its solution to the problem of initiating a powerful launch from water; thus, what was relevant was the detail that multiple, relatively small missiles are launched one at a time. This focus on individual launches helped suggest a new way of looking at the problem (Kolodner & Wills, 1993b).

3.4 Situation

Situation is our fourth aspect of thought. Design does not typically occur in a vacuum. Rather, designers usually try to experiment with their design (e.g., a mock-up, simulation, prototype, or partial construction) in a real-world situation (e.g., the typical operating environment, a potential maintenance situation, a worst-case scenario). This provides concrete feedback that can refine the problem specification to require any positive features noticed and to prohibit any flaws that were detected. At the same time, the evolving specification can be used to reinterpret entities in the environment and realize their relevance to the problem at hand.

Designers operate in a rich context of ideas, which are not only recalled and adapted from previous experiences, but also recognized in the current external environment. (That is, the environment can be a source of inspiration, in addition to knowledge and experiences recalled.) The continual elaboration and reformulation of the problem and desired solution primes the designer to recognize good ideas when they are stumbled upon. Problem redescription often enables the designer to overcome functional fixedness and notice new, alternative functions and uses for common design pieces. This leads to insights into new ways of solving pending problems (thus facilitating serendipity).

For example, at one point in the ME design project, the students were considering using a spring launch device, but had the problem that the springs bent when compressed. After generating, simulating, and critiquing a few proposals, they augmented their specification to require that each spring be enclosed in a collapsible tube. However, they could not immediately think of anything that could serve as a collapsible tube, so they temporarily gave up on designing the launch mechanism. Later, as they were looking for protective egg cushioning material, they came across toilet paper holders and immediately recognized them as the collapsible tubes they needed to keep springs straight (Wills & Kolodner, 1994b). By playing with the springs, noticing problems, and suggesting fixes, the designers formed a specific, concrete description of what they needed. This description was used to reinterpret the paper holder when it was seen and to recognize its additional function of preventing springs from bending upon compression.

Being situated facilitated the designers' discovery by bringing to their attention objects that could solve their problem without requiring the objects to be recalled as relevant solutions. Playing with the springs in a concrete situation also provided feedback to help the designers elaborate and refine their description of what they needed. The designers became immersed in the problem – redescrining it and viewing it from multiple perspectives,

considering, comparing, and critiquing several options – so that when a relevant solution was spotted, the way it fit into the problem was immediately discerned.

The importance of becoming immersed in the problem situation is implicitly acknowledged by Boden when she interrupts Chapter 4 to encourage the reader to temporarily stop reading and to play the necklace-building game. She suggests that the reader practice building necklaces (with pencil and paper), play around with the rules, record any interesting things that are noticed, etc. Although Boden does not analyze why this is so important, constructing specific necklace-building situations does provide feedback that can help the reader understand the problem constraints, their implications, and ways of modifying them.

3.5 Strategic Control

Finally, the fifth aspect of thought is the strategic control of inferences. Designers must make many decisions over the course of a design: which idea to elaborate or adapt next, which constraint to relax, how to set priorities. They also move between various tasks, subproblems, and design processes in a flexible and highly opportunistic manner.

We observed a variety of strategic control heuristics used by our ME designers. Some were opportunistic. An example is letting extremes distract. When an alternative was proposed that satisfied some desired criteria extremely well compared to the other alternatives, our designers directed their efforts toward elaborating that alternative (Wills & Kolodner, 1994b). They optimistically suspended criticism or discounted the importance of criteria or constraints that were not satisfied as well. Suspending criticism during brainstorming is a common strategic ideation technique which involves taking a cognitive risk. A similar mechanism is seen in creative interpretation, in which the reader must suspend disbelief in unfamiliar aspects of a story in order to understand it (see below). Sometimes, as constraints are relaxed or placed at a lower priority, an opportunity to reformulate the problem is revealed (Kolodner & Wills, 1993b). Noticing invariants (Kaplan & Simon, 1990), as well as anomalies, can also aid in understanding a problem and reveal ways of redescribing it.

Some strategic control heuristics are more deliberate, based on reflection. For example, one heuristic our designers used was to try quick, easy adaptations of a proposed solution first before stepping back and reformulating the problem or relaxing constraints (Wills & Kolodner, 1993a, 1994a). Other deliberate heuristics include making non-standard substitutions (Kolodner, 1994; Kolodner & Penberthy, 1990), applying adaptation strategies in circumstances other than the ones they were meant for (Kolodner, 1994; Navinchandra, 1992), merging pieces of separate solutions with each other in nonobvious ways (Kolodner, 1994; Kolodner & Penberthy, 1990), and goal-directed inferential control (Nersessian, in press; Ram, 1991; Ram & Hunter, 1992).

Often, creativity arises when a set of “normal” strategies are applied to a situation in which a run-of-the-mill solution is not immediately forthcoming and the control heuristics allow the reasoner to devote more resources to the problem, looking further and further afield for possible knowledge and strategies until something results in a creative solution. Examples include a problem reformulation that takes several steps; an analogy to a far-off case or model; an analogy from a hybrid analog constructed incrementally from more than one source; a strategy imported from a different problem-solving culture; an unexpected and novel opportunity afforded to the reasoner by virtue of an unusual task context. Many

of these could happen during "ordinary thought," but most thought does not allow enough leeway to look that far or to play with ideas for that long or it does not occur in a context that affords such an opportunity.

4 Beyond the Thinkable

Based on this view of creative thought, we offer a very pragmatic definition of the *normal* search space. It is not the deductive (or other) closure of everything that is known – an inherently uncomputable concept. Rather it is the *space of the thoughts one would usually explore in a pragmatic context*. There may be cases where important possibilities are outside the space of theoretically conceivable thoughts. (Perhaps rings of carbon atoms could never arise within the chemical theory prevailing at the time Kekule tackled benzene.) But, in other cases, thoughts that are within the theoretical space are nevertheless pragmatically inconceivable (e.g., the discoveries made by Swanson's (1990) program which are nevertheless H-creative). In creative individuals, even the usual search space may be interestingly different or expanded so as to provide the basis for creative thought using the very same mechanisms that on other occasions would produce more mundane thoughts.

Consider, for example, the problem of reading a science fiction story. Although creativity is usually thought of in the context of problem-solving or inventive tasks, we believe that creativity is an essential and ubiquitous component of other kinds of reasoning tasks as well, including explanatory and comprehension tasks. In point of fact, all these tasks involve understanding. Reading science fiction stories requires what we call *creative understanding*, in which the reader must learn enough about an alien world in a short text in order to accept it as the background for the story and simultaneously must understand the story itself. Creative understanding requires the extrapolation, modification, or extension of existing concepts and theories to invent new ones (Moorman & Ram, 1994a, 1994b; Ram, 1993). The extrapolation is constrained by the content of the story, by the system's existing concepts and theories, and by the requirements of the reading and understanding task.

As an example, consider the following short story, *Men Are Different* by Alan Bloch (1963).

I'm an archaeologist, and Men are my business. Just the same, I wonder if we'll ever find out about Men – I mean *really* find out what made Man different from us Robots – by digging around on the dead planets. You see, I lived with a Man once, and I know it isn't as simple as they told us back in school.

We have a few records, of course, and Robots like me are filling in some of the gaps, but I think now that we aren't really getting anywhere. We know, or at least the historians say we know, that Men came from a planet called Earth. We know, too, that they rode out bravely from star to star; and wherever they stopped, they left colonies – Men, Robots, and sometimes both – against their return. But they never came back.

Those were the shining days of the world. But are we so old now? Men had a bright flame – the old word is "divine," I think – that flung them far across the night skies, and we have lost the strands of the web they wove.

Our scientists tell us that Men were very much like us – and the skeleton of a Man is, to be sure, almost the same as the skeleton of a Robot, except that it's made of some calcium compound instead of titanium. Just the same, there are other differences.

It was on my last field trip, to one of the inner planets, that I met the Man. He must have been the last Man in this system, and he'd forgotten how to talk – he'd been alone so long. I planned to bring him back with me. Something happened to him, though.

One day, for no reason at all, he complained of the heat. I checked his temperature and decided that his thermostat circuits were shot. I had a kit of field spares with me, and he was obviously out of order, so I went to work. I pushed the needle into his neck to operate the cut-off switch, and he stopped moving, just like a Robot. But when I opened him up he wasn't the same inside. And when I put him back together I couldn't get him running again. Then he sort of weathered away – and by the time I was ready to come home, about a year later, there was nothing left of him but bones. Yes, Men are indeed different.

In order to understand this story, the reader must infer that the narrator is a robot, that robots are the dominant lifeform in the future, that humans have practically died out, that robots are capable of making factual errors such as the ones that the narrator made, and so on. The reader must construct an appropriate model of this world, and interpret the story with respect to this model even as the model evolves. The reader must also be willing to *suspend disbelief* (Corrigan, 1979) to understand concepts which do not fit into a standard world view. This is another example of a strategic control mechanism that requires a willingness to take a cognitive risk.

In *Men Are Different*, robots, which in the real world are physical objects used as tools in manufacturing, are conceptualized as independent volitional agents. The new concepts are constructed by merging and extending the existing concepts representing human agents and robotic artifacts, resulting in a novel view of the situation at hand (Moorman & Ram, 1994a). The reader must adopt this view to build an appropriate story model. Interestingly, the irony in this story derives from the fact that the robot in the story performs what one might view as the reverse inference: conceptualizing the man as a physical object to be repaired in a manner that one might use to repair a physical robotic device (Moorman & Ram, 1994b).

It would, of course, be unreasonable to assume a special purpose "meta-search space" generator for science fiction story understanding. The creative understanding processes required to read *Men Are Different* are not unique to science fiction stories; understanding any fictional story requires similar kinds of processing. The same is true of nonfictional stories as well as unfamiliar real-world scenarios, although the types and degree of conceptual modifications required may be different.

Thus, reading a science fiction story is presumably accomplished within the same type of search space and using the same set of reading and comprehension operators as reading a mundane narrative. The example illustrates that these ordinary operators and processes can take the reasoner out of the space that would usually be explored. In fact, situations like this show just how fluid the movement is from the usual to the unusual.

The question, of course, is how the search space comes to be expanded to facilitate creative thought using ordinary mechanisms. If normal traversal of a search space depends on knowledge, inferential methods, and control methods, then interesting paths may result from modifying any of these three components. Most obviously, transformations of basic knowledge (e.g., conceptual change) can yield new results. But application of new inferential methods can also produce novelty; for example, adopting a heuristic from a different task context, such as an architect adopting the engineer's heuristic of "incorporate the obstacle." Finally, differences in control methods will produce differences in results; consider methodological differences between scientists, such as the willingness to take cognitive risks, the willingness to explore a "silly" idea, the ability to evaluate and prune unlikely candidates. For example, we would rate AM+Lenat as a creative combination even though AM by itself was not. Analysis of the task and situation influences the knowledge, inferential methods, and control strategies that are available.

5 Constructive Modeling

Reading and understanding *Men are Different* requires the invention of a system of concepts and theories that represent a sentient, humanoid robot, through the extension of one's prior understanding of multiple concepts, such as volitional agents, men, and industrial robots (Moorman & Ram, 1994a). In creative design, too, new conceptual structures are formed from multiple sources. Problem descriptions are incrementally elaborated and reformulated, typically by analogy to pieces of several similar problems. New design ideas are generated by combining several ideas from experiences with existing devices. The behavior of a proposed design is predicted, simulated, and visualized based on multiple pieces of knowledge of how related devices or design pieces work.

These are everyday instances of the constructive modeling process we have found to be central in significant scientific discoveries throughout the history of the sciences. For example, it figures centrally in the development of the field representation of electromagnetic forces by Michael Faraday and James Clerk Maxwell. Here we will illustrate our points by looking briefly at Maxwell's derivation of the electromagnetic field equations (Maxwell 1890). The Maxwell case reinforces Boden's contention that even in instances of H-creativity, explaining the episode demands an analysis of P-creativity.

This case shows constructive modeling to be a dynamic process involving analogical and visual modeling as well as thought experimentation (mental simulation) to create sources where no direct analogy exists (Clement, 1989; Nersessian 1992, 1993, in press; Nersessian & Greeno, in process). What distinguishes this process from the computational models of analogical reasoning Boden discusses is that they employ cases where the analogical base is ready to hand. Further, although Boden does note the importance of visual representation in some instances of analogy, neither she nor the computational models she discusses attempt to integrate it into their accounts. Indeed, we believe the constructive modeling processes identified in the Maxwell case show the need for an integrated account of analogy, visual representation, and mental modeling for understanding creative thinking.

Finally, this case points to something missing entirely from Boden's analysis. The social context is crucial to understanding a creative episode in science - and we presume in more ordinary cases, too. Maxwell's location in Cambridge led to his training as a mathematical

physicist. This determined the nature of the theoretical, experimental, and mathematical knowledge and the methodological practices with which he formulated the problem and approached its solution. The work of Faraday and William Thomson (later, Lord Kelvin) contributed to these as well. Continental physicists working on electromagnetism at the same time employed quite different practices and drew from fundamentally different mathematical and physical representational structures. These kinds of social factors can be figured into the account without our being required to produce a computational model of culture.

Maxwell's constructive modeling process provides a good example of an instance in which all five of the aspects of creative thinking we have been discussing are employed. He used multiple knowledge domains and informational formats, in the context of solving a complex problem within specific cognitive and social situation. Maxwell exercised strategic control continually to evaluate the models and the inferences he drew from them, and to integrate the solutions to the sub-problems into a consistent mathematical representation. The modeling process involved adjusting multiple constraints drawn from

- the physics of elastic fluids,
- experimental data on electricity and magnetism,
- Faraday's hypotheses about the lines of force that form when iron filings are sprinkled around magnets and charged matter (Faraday, 1835-55),
- Faraday's visual lines of force model (shown in Figures 1a and 1b), accounting for continuous transmission and interconversion of forces (Maxwell, 1890, vol. 1, pp. 155-229),
- Faraday's interlocking curves model (shown in Figures 2a and 2b), representing the dynamical balance between electricity and magnetism (Maxwell, 1890, p. 194n), and
- William Thomson's hypothesis of rotational motion of magnetism and his analogies, and mathematical equations (Lamor, 1937).

Maxwell's goal (Maxwell, 1890, vol. 1, pp. 451-513) was to provide a unified representation of the continuous transmission of electric and magnetic forces that he hoped would encompass optical phenomena as well. The full model is an imaginary hybrid construction that integrates physical and mathematical constraints from two analogical source domains – continuum mechanics (fluids, elastic media, etc.) and machine mechanics – with constructs from magnetism and electricity. Unlike the cases customarily considered in the literature on analogy, where an existing problem solution in the source domain is transferred to the target domain, in this case, the source and target domains interact to create and modify a series of constructed models that become the objects with which Maxwell reasoned (Nersessian, in press; Nersessian & Greeno, in process). Further, reasoning with the models demands that they provide simulations and thus be animated in a manner similar to thought experiments (Nersessian 1993). In the text itself, Maxwell provided an extensive set of instructions for how the reader should visualize and animate the models.

Maxwell's model construction proceeded as follows. Maxwell first constructed a primitive model (Figure 3a) consistent with the constraints discussed above: a fluid medium composed of elastic vortices and under stress. With this form of the model he was able

to provide a mathematical representation for several magnetic phenomena. Analyzing the relationships between current and magnetism required alteration of the model. We can see in Figure 3a that all the vortices are rotating in the same direction, which means that since they touch, friction is produced and they will eventually stop. Mechanical consistency, thus, requires the introduction of "idle wheels" (as in machine gears) surrounding the vortices, and Maxwell argued that their translational motion could be used to represent electricity. Figure 3b shows a cross section of the hybrid model. For the purposes of calculation, Maxwell now had to make the elastic vortices into rigid pseudospheres. We can see how the imaginary system provides a mechanical interpretation for electromagnetism: motion of the particles creates motion of the vortices and vice versa. In this model, as was known experimentally, electric current produces magnetic effects and changes in magnetic effects produce current. Using the model, he derived mathematical equations to represent these relationships.

It then took Maxwell nine months to figure out how to represent the final – and most critical – piece of the problem: electrostatic actions. He found that if he made the vortices elastic and identified electrostatic polarization with elastic displacement, he could calculate the wave of distortion produced by polarization. That is, adding elasticity to the model enabled him to show that electromagnetic actions are propagated with a time delay, i.e., they are field actions and not Newtonian actions at a distance. At this point, we have a fully mathematized representation of the electromagnetic field. There are significant sign "errors" in this part of Maxwell's analysis, but Nersessian (1984, in press) has argued that all but one (a minor substitution error) can be seen not to be errors when we view him as reasoning via the constructed model.

This case study illustrates that it was through a process of embodying physical and mathematical constraints in a series of constructed models and reasoning about and with these that Maxwell generated the field equations for electromagnetism – an *historically and individually* creative process.

6 Summary and Conclusions

Inference and the control of inference, knowledge representation and representational change: these are the main interrelated pieces of the creativity puzzle. Each relies heavily on episodic and semantic memory. Together, they fit into a model of reasoning that is recognizable as (but looser than) case-based reasoning. A creative individual is one in whom these factors combine to form a search space – a repertoire of thoughts – that is different from the usual and contains many creative ideas waiting to be constructed. Of course, the search space can only be explored in the context of a task or problem and a specific situation; thus, the repertoire is defined pragmatically, and serendipity (as Boden points out) plays an important role.

In a specific individual, more creative thoughts will likely result when these pieces come together in a novel way to yield an unexplored and unexpected path through the search space. Creativity, as Boden points out, is not an all-or-none phenomenon. Every new thought is creative to some extent. Every new thought results from those same processes that, on occasion, produce results we value as creative. The more the search space is varied in a given context (through representational change, novel inferential methods, or strategic

control heuristics), the more creative the resulting thoughts are likely to be. Over time, an individual may become more expert as he or she acquires (or reformulates) knowledge, reasoning strategies, and methodologies that change the search space or how it is explored.

The framework we have sketched here is broadly compatible with Boden's, but is more specific in its suggestions for integrating multiple types of interacting and interactive processes in a task context. In accounting for creativity, we emphasize issues of control and the role of experience (or cases). By focusing on how mental activity is directed towards a task in some situation, we ensure that the resulting theory addresses pragmatic issues in thinking and control of thinking. As Boden would require, our approach is computational. We believe, in fact, that the greatest contribution of *The Creative Mind* is the clear case it presents for the legitimacy of computational theories of creativity. Boden leads the reader to an understanding of that goal, and, having framed the question, suggests how research might proceed towards a meaningful answer.

7 Acknowledgments

This paper is based in part on research by Kenneth Moorman, who also provided us with helpful suggestions for improving this paper. We are grateful to our editors Mark Stefik and Stephen Smoliar for their many insights and helpful suggestions for improving this review. This work has been supported in part by the Advanced Research Projects Agency, monitored by ONR under contract N00014-91-J-4092, by NSF Grant No. IRI-8921256 and ONR Grant No. N00014-92-J-1234, by NSF Scholars Awards DIR8821442 and DIR9111779, and by the Georgia Institute of Technology. All views expressed are those of the authors.

8 Bibliography

- Bloch, A. (1963). "Men are Different" in *50 Short Science Fiction Tales*, Asimov, I. and Conklin, G. (eds.), New York: MacMillan Publishing Co.
- Boden, M.A. (1990). *The Creative Mind: Myths and Mechanisms*. London: Weidenfeld and Nicolson Ltd. (Expanded edition, New York: BasicBooks, 1992.)
- Carey, S. (1985). *Conceptual Change in Childhood*. Cambridge, MA: MIT Press.
- Clement, J. (1989). Learning via Model Construction and Criticism. In Glover, G., Ronning, R., & Reynolds, C. (eds.), *Handbook of Creativity: Assessment, Theory and Research*, pp. 341-381. New York: Plenum, 1989.
- Corrigan, R.W. (1979). *The World of the Theatre*. Glenview, IL: Scott, Foresman and Co.
- Dietterich, T.G. (1986). Learning at the Knowledge Level. *Machine Learning*, Vol. 1, pp. 287-316.
- Faraday, M. (1835-55). *Experimental Researches in Electricity*. Reprinted, New York: Dover.
- Jones, E. (1992). Brainstormer: A Model of Advice Taking. PhD Thesis, Yale University.

- Kaplan, C. & Simon, H. (1990). In Search of Insight. *Cognitive Psychology*, Vol. 22, pp. 374-419.
- Kolodner, J. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan-Kaufmann.
- Kolodner, J.L. (1994). Understanding Creativity: A Case-Based Approach. In S. Wess, K.D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, selected papers from the *First European Workshop on Case-Based Reasoning*. Kaiserslautern, Germany. November 1993. Springer-Verlag.
- Kolodner, J. & Penberthy, T. (1990). A Case-Based Approach to Creativity in Problem Solving. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. August.
- Kolodner, J. & Wills, L. (1993a). Case-Based Creative Design. In *AAAI Spring Symposium on AI and Creativity*. Stanford, CA. March 1993. Reprinted in *AISB Quarterly*, special issue (no. 85) on AI and Creativity, edited by Terry Dartnall, Autumn 1993.
- Kolodner, J. & Wills, L. (1993b). Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design. In *AAAI Case-Based Reasoning Workshop*, (pp. 19-25).
- Kuipers, B.J. & Byun, Y.-T. (1991). A Robot Exploration and Mapping Strategy based on a Semantic Hierarchy of Spatial Representations. *Robotics and Autonomous Systems*, 8(1-2): 47-63
- Lamor, J., ed. (1937). *The Origins of James Clerk Maxwell's Electric Ideas*. Cambridge: Cambridge University Press.
- Maxwell, J.C. (1890). *The Scientific Papers of James Clerk Maxwell*, W.D. Niven, ed. Cambridge: Cambridge University Press.
- Moorman, K. & Ram, A. (1994a). A Model of Creative Understanding. To appear in the *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. Seattle, Washington.
- Moorman, K. & Ram, A. (1994b). Integrating Creativity with Reading: A Functional Approach. To appear in the *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. Atlanta, GA.
- Navinchandra, D. (1992). Innovative Design Systems: Where are we, and where do we go from here?. Parts I and II. *The Knowledge Engineering Review*, 7(3) pp. 183-213 and 7(4) pp. 345-362.
- Nersessian, N. (1984). *Faraday to Einstein: Constructing Meaning in Scientific Theories*. Dordrecht: Kluwer Academic Publishers.
- Nersessian, N. (1992). How Do Scientists Think? Capturing the Dynamics of Conceptual Change in Science. In *Volume XV: Cognitive Models of Science* of R Giere, Ed., *Minnesota Studies in the Philosophy of Science* 15. Minneapolis: University of Minnesota Press.

- Nersessian, N. (1993). In the Theoretician's Laboratory: Thought Experimenting as Mental Modeling. In *Proceedings of the 1992 Meeting of the Philosophy of Science Association*, Vol. 2, D. Hull, M. Forbes, and K. Okruhlik Eds. East Lansing Michigan: Philosophy of Science Association.
- Nersessian, N. (in press). Abstraction via Generic Modeling in Concept Formation in Science. In Cartwright, N. & Jones, M.R. (eds). *Idealization in Science*. Amsterdam: Editions Rodopi.
- Nersessian, N. & Greeno, J. (in process). Dynamic mental modeling in scientific reasoning, manuscript.
- Newell, A. (1982). The Knowledge Level. *Artificial Intelligence* 18(1): 87-127.
- Ram, A. (1991). A Theory of Questions and Question Asking. *The Journal of the Learning Sciences*, 1(3-4): 273-318.
- Ram, A. (1993). Creative Conceptual Change. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pp. 17-26, Boulder, CO.
- Ram, A. (1994). AQUA: Questions that Drive the Explanation Process. In R.C. Schank, A. Ram, and C.K. Riesbeck (eds.), *Inside Case-Based Explanation*, Chapter 7. Lawrence-Erlbaum Associates.
- Ram, A. & Hunter, L. (1992). The Use of Explicit Goals for Knowledge to Guide Inference and Learning. *Applied Intelligence*, 2(1): 47-73.
- Ram, A. & Leake D. (1991). Evaluation of Explanatory Hypotheses. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pp. 867-871, Chicago, IL.
- Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press.
- Schank, R. (1986). *Explanation Patterns: Understanding Mechanically and Creatively*. Hillsdale, NJ: Lawrence-Erlbaum Associates.
- Swanson, D. (1990). Medical Literature as a Potential Source of New Knowledge. *Bull. Med. Libr. Assoc.*, 78(1):29-37.
- Thagard, P. (1992). *Conceptual Revolutions*. Princeton, NJ: Princeton University Press.
- Turner, S.R. (1994). *MINSTREL*. Lawrence-Erlbaum Associates. Forthcoming.
- Wills, L.M., & Kolodner, J.L. (1994a). Towards More Creative Case-Based Design Systems, to appear in the *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. Seattle, Washington.
- Wills, L.M., & Kolodner, J.L. (1994b). Explaining Serendipitous Recognition in Design, to appear in the *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. Atlanta, GA.

POWERS OF OBSERVATION IN CREATIVE DESIGN

JANET L. KOLODNER AND LINDA M. WILLS

*College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 894-3285
Fax: (404) 853-9378
jlk@cc.gatech.edu*

Abstract. Being perceptive is a trait highly valued in scientific and engineering professions. What a scientist or engineer notices while considering a problem, evaluating alternatives, or interpreting data has a profound impact on how a problem is viewed and solved.

This paper focuses on the processes we believe underlie being perceptive: (1) *preparation* – becoming attuned to salient or important features; (2) *assimilation* – detection and exploration of patterns (invariants) as well as anomalies; and (3) *strategic control* – heuristic strategies for exploring and pursuing the implications of what has been observed.

It shows that these processes help to explain important issues of focus in creative design and play an integral role in characteristic activities within creative design. These include problem reformulation, the emergence of properties and constraints on the solution, and the ability to incorporate into the design experimental feedback from the environment and from experiences with prototypes and previous designs. The paper presents a computational model incorporating these ideas, which we have implemented in a system called Improviser.

This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234.

*To appear in
~~Submitted to~~ Design Studies, Special Issue on Design Cognition and
Computation, edited by Rivka Oxman.*

1. Introduction

Being perceptive – noticing the relevance of features observed to a current or pending problem of interest – is a trait highly valued in scientific and engineering professions. What a scientist or engineer notices while considering a problem, evaluating alternatives, or interpreting data has a profound impact on how a problem is viewed and solved. We consider three aspects of creative design in which perceptive observation plays an essential role: problem reformulation, augmenting the teleology of design pieces, and criteria emergence.

These aspects are illustrated with examples we collected during an exploratory study in which we observed a team of four designers in a seven-week mechanical engineering (ME) design project (Kolodner and Wills, 1994, 1993a, 1993b). The design task was to build a device to quickly and safely transport as many eggs as possible from one location to another. The device could be constructed from any set of materials, as long as it satisfied a set of size, weight, and cost restrictions.

Our analysis suggests that three key processes underlie being perceptive: preparation, assimilation, and strategic control. Based on these ideas, we have developed and implemented a computational model that uses case-based reasoning (Kolodner, 1993) as a theoretical framework.

To better understand the phenomena we are interested in, consider how perceptive observation plays a role in the following three characteristic activities of creative design.

PROBLEM REFORMULATION: NOTICING IMPLICIT ASSUMPTIONS

In design, there are often default assumptions about the constraints of the problem. These come from previous solutions that have been applied to the problem or similar problems. In effect, the solutions provide implicit constraints on what is held constant in the space of solutions that can be considered. What is needed to break out of this rut is for the designer to realize that a feature that happens to be constant across previous solutions can actually vary. That is, a new design variable (Gero, 1990; Gero and Maher, 1993) is added to the problem.

One way that we have observed this happening is for designers to *notice that a proposed solution violates a default assumption* (or implicit problem constraint). This can dramatically change the problem description, as in the following example.

In the ME design project, the designers were thinking of various ways of

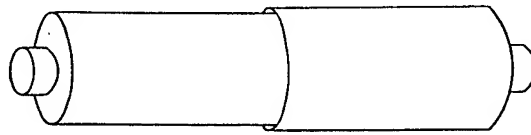


Figure 1. Toilet paper holder recognized as collapsible tube for supporting a spring.

launching a device containing all the eggs that needed to be transported. At one point, the designers were considering launch mechanisms for moving the eggs out of a child's wading pool filled with water (one of the possible launch locations). One designer said that the eggs would need to be launched like a missile from a submarine and acted out the launch with a pen. This reminded another designer that submarines launch multiple, relatively small missiles one at a time. Up until this point, all the solutions proposed treated the eggs as a single group that was moved all at once.

Noticing this difference led to reformulating the problem from one of moving all eggs as a group to moving eggs individually. In essence, what was assumed to be a constant across all solutions (launch one group of all eggs) became a design variable whose assignment could be explicitly decided upon (some number of launches each launch some number of eggs).

SERENDIPITOUS RECOGNITION: AUGMENTING TELEOLOGY

What is noticed as being relevant not only influences what new design variables emerge, but it can also determine how an object is used and what functional properties are assigned to it. Creative designs often make use of design pieces in novel ways, overcoming fixation on their usual purpose or functional role.

An example of this type of creative reuse occurred in the ME design project. Our designers were considering using a spring launching device and went to a home improvement store to look into materials. While comparing the strengths of several springs by compressing them, they noticed that the springs tended to bend. One designer wrapped a hand around the spring to hold it straight as it was compressed and said the springs would each need to be enclosed in a tube to keep them from bending. Another added that the tube would need to be collapsible (to compress with the spring). The designers could not think of an existing collapsible tube and did not want to build one due to time pressure. They gave up on the springs and started thinking about egg protection. During

their search for protection material, they walked through the bathroom section of the store, where they saw a display of toilet paper holders (see Figure 1). They immediately recognized them as collapsible tubes which could be used to support the springs.

How the toilet paper holder was perceived and what properties were seen in it allowed it to be seen as being able to provide a function besides holding a roll of paper. It is assigned an additional function of laterally supporting the internal spring, while flexibly allowing it to compress and expand longitudinally.

We call this process *serendipitous recognition*: seeing solutions to pending design problems in objects in the surrounding environment (Wills and Kolodner, 1994a). It often involves noticing that ordinary design pieces have new functions or purposes in addition to their ordinary ones.

CRITERIA AND CONSTRAINTS EMERGENCE: NOTICING ANOMALIES

Noticing anomalies in experimental data is well-recognized as a powerful impetus of conceptual change in scientific discovery, where a new model or representation can result from the reinterpretation of data (Nersessian, 1992; Vosniadou, *et al.*, 1992). In design, too, noticing unexpected properties or behaviors in experimental prototypes is a key impetus for emerging evaluation criteria and constraints.

In the bending spring example described above, for instance, the designers refined and elaborated the specification of their spring launch mechanism. This evolution was driven by quickly simulating partial, experimental prototypes, such as compressing a spring between a thumb and forefinger and wrapping a hand around the spring being compressed. As unexpected consequences occurred, such as the spring bending, additional constraints were added to the specification.

This gradual emergence of evaluative criteria (Navinchandra, 1991), constraints in general (Prabhakar and Goel, 1992), and the relative priorities among them is a key characteristic of creative design. Creative design involves deriving new constraints and priority structures that improve or go beyond those stated in the original problem description.

PROCESSES UNDERLYING PERCEPTIVE OBSERVATION

This paper focuses on the processes we believe underlie being perceptive.

1. **preparation** – becoming attuned to salient or important features;

2. **assimilation** – detecting patterns or anomalies and exploring them; being attracted to or intrigued by invariants or anomalies that are noticed.
3. **strategic control** – heuristic strategies for exploring and pursuing the implications of what has been observed. This includes being distracted by incongruities, extremes, and unexpected similarities.

We first describe the two complementary processes: preparation and assimilation. We then discuss what guides and focuses these processes and the strategic control that is involved. Throughout this discussion, we point out computational mechanisms that are relevant to modeling these processes. We then present an implemented system, called IMPROVISER, that brings these mechanisms together within a framework rooted in case-based reasoning (CBR) (Kolodner, 1993). The architecture of this system is first described and then a detailed scenario is given to demonstrate how it models perceptive observation in the context of a creative design situation.

2. Preparation

To notice that an object has properties or behaviors that make it useful for a new purpose or that an unexpected behavior of an experimental prototype is interesting, a reasoner must learn what features to pay attention to in the problem solving environment.¹ The designer needs to learn what to focus on and what is relevant.

Researchers studying insight (e.g., (Wallas, 1926) and (Seifert, *et al.*, 1995)) refer to this learning process as *preparation*. It involves becoming immersed (Goel and Pirolli, 1989) in the problem, redescribing it and viewing it from multiple perspectives, redescribing it in familiar terms, and considering, comparing, and critiquing several solution options.

During preparation, the specification of the problem evolves. It typically starts out ambiguous, incomplete, contradictory, and underconstrained. Through a confluence of interacting processes, a new problem specification emerges which is more detailed and concrete. It has more clearly defined and consistent constraints. Constraints that initially were vague and abstract are refined to more operationalized conditions that are concrete, specific, and efficiently recognizable. At the

¹By "environment" we mean not only the external surroundings of the designer, but also the internal context of memories recalled and ideas, data, etc., about which the designer is currently reasoning.

same time, constraints that are too specific are generalized and made more abstract. New constraints and criteria emerge.

What are the interacting processes? There is not a single specification evolution mechanism, but several evolutionary mechanisms that interact. Some examples of deliberate evolution mechanisms are the following:

- reinterpretation of an idea in terms of a different but familiar idea (e.g., (Jones, 1992) shows how this can lead to useful problem reformulations which facilitate the operationalization of abstract advice (in the form of proverbs) during planning situations);
- visualization, mental simulation, and thought experimentation, which are useful in evaluating and elaborating ideas, and in reformulating problems in design (Kolodner and Wills, 1993b) and scientific reasoning (Nersessian, 1992; Nersessian, 1993);
- constraint relaxation and substitution, which are useful in problem reformulation and elaboration (e.g., (Moorman and Ram, 1994a) show how new concepts can be formed or understood, while reading science fiction stories, by systematically tweaking constraints on known, familiar objects);
- relaxing constraints during memory search, which facilitates problem reformulation and retrieval (e.g., (Turner, 1994) calls this *imaginative retrieval* and shows how it can be used to retrieve ideas for writing short stories).

Specification evolution is not autonomous or independent. Evolutionary processes are interleaved with and interact closely with the processes of recalling potential solutions or similar past problems and evaluating proposed solutions. As proposals are considered, evaluation detects contradictions and ambiguities in the specification that prevent a proposed solution from being definitively accepted or rejected. The resolution of these questions, contradictions, and ambiguities serves to refine, augment, and reformulate the design specification. Reasoning about the causes of negative or positive design features that are identified during evaluation leads to the addition of constraints to the specification that prohibit or require these features (Bhatta, *et al.*, 1994). In addition, proposed solutions often directly remind designers of issues to consider (Domeshek and Kolodner, 1993). The problem and solution co-evolve (Fischer, 1993).

The evolutionary processes involved in preparation depend heavily on the retrieval and manipulation of past experiences. It is not surprising that several of the computational models that have been developed and studied within the theoretical

framework of case-based reasoning are relevant to modeling preparation in creative design.

In particular, evaluation includes outcome determination, which may be achieved by simulation or by *case-based prediction* (Kolodner and Penberthy, 1990; Navinchandra, 1991).

Also, problem interpretation and reformulation correspond closely with the CBR process of *situation assessment* – redescribing a problem in the vocabulary of a memory's indexing scheme. Investigating a problem in depth makes available a large set of relevant cues for retrieval. Generating multiple ways of describing a problem provides several different contexts for specifying what would be relevant, if remembered. Research on indexing has found that it is the combination of setting up a context for retrieval and having already interpreted something in memory in a similar way that allows retrieval. When some case or piece of knowledge is entered into memory, it is not always possible to anticipate how it might be used. Situation assessment processes aim to bridge that gap by helping to redescribe a new problem in a way that is similar to something seen before. Research into situation assessment and problem reformulation (e.g., in CASEY (Koton, 1988), CYRUS (Kolodner, 1983), MINSTREL (Turner, 1994), and BRAINSTORMER (Jones, 1992)) show various ways it can be done. In addition, Sycara and Navinchandra (Navinchandra, 1992; Sycara and Navinchandra, 1989) summarizes several related *index transformation and elaboration* techniques that have been developed.

Research on *predictive encoding* (Patalano, *et al.*, 1993; Hammond, 1989) is also relevant. The key idea is that when a reasoner (in this case, a planner) reaches an impasse (e.g., a resource needed for executing a plan is not available), the reasoner suspends the task (or plan), saving away an encoding of what would be needed to allow the task to resume. These encodings are in the form of memory indexing structures. They associate the suspended task goal with features that would be in the environment if it were favorable to achieving the goal. For example, suppose a planner were in the middle of making breakfast and found out that there was no orange juice. The planner would suspend the goal of having orange juice and index it with features that would exist in an environment in which obtaining orange juice were possible (e.g., the planner's location is a grocery store).

This work is a step toward explaining preparatory processes. However, it is studied in the context of opportunistic planning situations in which there are *standard* solutions to the problems at hand. Therefore, the choice of what features to

pay attention to (i.e., what features are used in the predictive encoding) is relatively straightforward. In creative design, on the other hand, suspended problems typically have unanticipated, nonstandard solutions. The choice of features to pay attention to that would signify an opportunity to resume and solve a pending problem must be dynamically derived. This occurs primarily during specification evolution as a description gradually emerges of what a potential solution to the problem would look like.

Throughout the process of specification evolution, the designer gradually becomes attuned to what is needed to solve pending subproblems and what evaluative criteria are relevant. The result is that when a relevant solution to the problem is spotted, the features that are necessary for achieving some new function or purpose are immediately discerned. Similarly, when experimental feedback is available, the designer knows which observations and data to focus on and which criteria to use to evaluate them.

3. Assimilation

As the designers in our ME study considered each proposed solution, they compared and contrasted it to previous solutions that had been proposed. They classified each and organized them according to differentiating properties, such as which subproblem it addressed (e.g., launch, transport, landing, protection), what energy source was involved (e.g., hydraulic, electric, combustion), and the shape of motion trajectory it took. The dimensions of comparison were usually the criteria and features the designers had become attuned to during preparation. The organization of ideas was continually expanded, shrunk, and restructured throughout the design.

In the process, distinctions between proposed solutions were pointed out and patterns of invariants across solutions were noticed. This was essential in deciding whether a proposed idea was novel or could be ignored. This often caused new evaluative issues to emerge as new criteria or dimensions of comparison were generated to distinguish seemingly identical ideas. Attention was drawn to features that were unusual or extremely good or bad compared to the properties of other proposed ideas (e.g., pine straw as cushioning material was "dirt cheap" and bottle rockets as launch devices were extremely dangerous and of questionable legality). A kind of "relative" evaluation was performed: each proposal is judged on how well it satisfies the specification constraints, compared to the other alternatives.

We refer to this process of organizing and detecting distinguishing features as *assimilation*. It includes “making sense of” not only proposed solutions, but also data and observations coming from the design environment, such as feedback from experiments with prototypes. As our ME designers tried out prototypes such as small-scale spring launch mechanisms and various materials for egg protection, they had sets of expectations about how these prototypes would work. As feedback from real-world experimentation was collected, it was compared to these expectations and to feedback from earlier experiments or recalled design cases. In the process, generalized patterns of behavior were learned and anomalies were detected.

A type of assimilation was also observed by (Kaplan and Simon, 1990) in an experiment to study insight problem solving. They observed that people working on the “mutilated checkerboard” problem² detected invariants across specific instances of the problem while trying various ways of solving it. This led to a new way of describing the problem that made it easy to solve.

A particularly useful model for studying assimilation is the *dynamic memory* model (Schank, 1982; Kolodner, 1983; Ram, 1994), which is one of the principle foundations of case-based reasoning. The key idea underlying dynamic memory is that remembering, understanding, and learning are all inextricably intertwined. The ability to determine where something fits in which what we already know (understanding) is a key part of being able to assimilate ideas and observations into our problem solving. This may involve a useful reinterpretation of something already in memory and can result in a new way of indexing it in memory.

A critical issue in modeling assimilation is where does the organizational structure, including initial expectations, come from that is used to compare, contrast, and categorize ideas and observations? There are several sources.

1. The **problem framework** provides one source of structure. This is the skeleton of the problem which holds its hierarchical decomposition into subproblems and the interactions among the problems. For example, the ME design project had four main subproblems (launch, transport, land, protect) each with several subproblems of their own. There were relationships between the subproblems, such as decision dependencies (e.g., the form of trajectory

²This is a famous insight problem in which two opposite corners of a checkerboard are removed and the problem is to either show that a set of dominoes, each covering two adjacent squares, can cover all the remaining squares of the checkerboard or prove that a complete covering is impossible.

initiated by the launch mechanism constrained the type of landing mechanism that was required). Designers frequently choose an already well-known framework (or generic case) for a problem and then fill it in. Reusing solution structures in this way allows designers to avoid recomputing useful compositions of design pieces. We call this process "framing a solution." The framework provides the glue holding the pieces of the design together. The creativity comes in filling in details and in dealing with inconsistencies when merging alternative pieces. Such framing occurs in domains, such as bridge design and engine design, where well-known frameworks exist and where constraints holding the pieces of the problem together are quick complex. In other domains, such as architectural design, creating the framework is a primary piece of the creative process. The problem framework is likely to be dramatically restructured as solutions are proposed and structural aspects are inferred from them.

2. The **problem specification** provides constraints and criteria that are a primary source of dimensions along which to compare and relatively evaluate solution alternatives. The constraints also provide a description of what behavior is expected of the solution.
3. While the specification *explicitly* defines what dimensions are relevant, the **solutions proposed** so far *implicitly* determine what is expected. They implicitly form a baseline for comparison. The submarine missile launch example presented earlier is an instance of this. In this case, the dimensions of comparison emerge or become explicit only when an idea is considered that "breaks out of the mold" by being significantly different from the rest.
4. **Prototype predictions** are another source of expectations used in comparison. When an experimental prototype is constructed and tried out, the designer usually has certain expectations about how it will behave. Or the designer might have questions about which of several possible behaviors it will exhibit. These expected behaviors form the basis for comparison when the prototype is actually run or simulated and the outcome is examined.

The process of assimilation involves four main subtasks. One is setting up the organizational structure, based on the sources described above. Another is placing an entity in the structure – finding out where it fits. A third is doing the compare and contrast to detect significant differences and expectation violations. (This task is straightforward for all the sources of expectations except for those implicit in

the solutions proposed so far (number 3 above), which remains an open issue.) The fourth task is to resolve the interesting anomalies, perhaps by reformulating or refining the problem resulting in a change to the organizational structure. This fourth task is guided by strategic control heuristics that explore the implications of the detected anomalies, as described in the next section.

The preparation and assimilation processes are complementary and feed on each other. Preparation helps determine what is expected. Resolving conflict during assimilation often changes the specification or view of the problem.

4. Control

Designers must make many decisions over the course of a design: which idea to elaborate or adapt next, which constraint to relax, how to set priorities. They also move between various tasks, subproblems, and design processes in a flexible and highly opportunistic manner. What a designer has noticed (e.g., incongruities, extremes, unexpected similarities) can have a profound impact on what the designer can and should do next.

We have identified several strategic control heuristics that guide a creative designer in deciding what to do next. Some strategic control heuristics are deliberate, based on reflection. For example, one heuristic our designers used was to try quick, easy adaptations of a proposed solution first before stepping back and reformulating the problem or relaxing constraints (Wills and Kolodner, 1994b). Other deliberate heuristics include making non-standard substitutions, applying adaptation strategies in circumstances other than the ones they were meant for, and merging pieces of separate solutions with each other in nonobvious ways (Kolodner, 1994). (See (Ram, *et al.*, 1995) for others.)

An example opportunistic control strategy is to let extremes distract. In the ME project, when an alternative was proposed that satisfied some desired criteria extremely well compared to the other alternatives, our designers directed their efforts toward elaborating that alternative (Wills and Kolodner, 1994a). They optimistically suspended criticism or discounted the importance of criteria or constraints that were not satisfied as well. Suspending criticism like this is a common strategic ideation technique which depends on willingness to take a cognitive risk.³

³A similar mechanism is seen in creative story interpretation, in which the reader must suspend disbelief (Corrigan, 1979) in unfamiliar aspects of a story in order to understand it (Moorman and

The full set of influences we've discovered so far on what actions a designer takes are the following.

- **Letting extremes distract** (described above).
- **Changes in the specification** – the appearance of a new specification of a subproblem, newly emerged constraints, or noticing that certain constraints are not refined or operational enough – can direct the designers attention to trying hard to satisfy the new constraints or refine them.
- **Current environment** – e.g., being at a hardware store provided our ME's with opportunities to buy, select, or test out materials, as well as search for potential solutions to pending problems.
- **Measures of progress** can be used to prioritize areas needing attention. For example, pending problems that are resisting solution point out areas that might need special focus and more resources. The importance or relative priority of the constraints and goals involved also factors in here. This depends on how tightly coupled the constraint or goal is with other subproblems. For example, the ME designers had to decide on the spring strength constraint early since it was holding up decisions on other subproblems, such as how much egg protection was going to be needed and how much breaking force would be needed to stop the device at the appropriate location.

Often, creativity arises when a set of "normal" strategies are applied to a situation in which a run-of-the-mill solution is not immediately forthcoming and the control heuristics allow the reasoner to devote more resources to the problem, looking further and further afield for possible knowledge and strategies until something results in a creative solution. Examples include a problem reformulation that takes several steps; an analogy to a far-off case or model; an analogy from a hybrid analog constructed incrementally from more than one source; a strategy imported from a different problem-solving culture; an unexpected and novel opportunity afforded to the reasoner by virtue of an unusual task context. Many of these could happen during ordinary thought, but most thought does not allow enough leeway to look that far or to play with ideas for that long or it does not occur in a context that affords such an opportunity.

5. Improviser: Experimental Implementation of Our Model

IMPROVISER

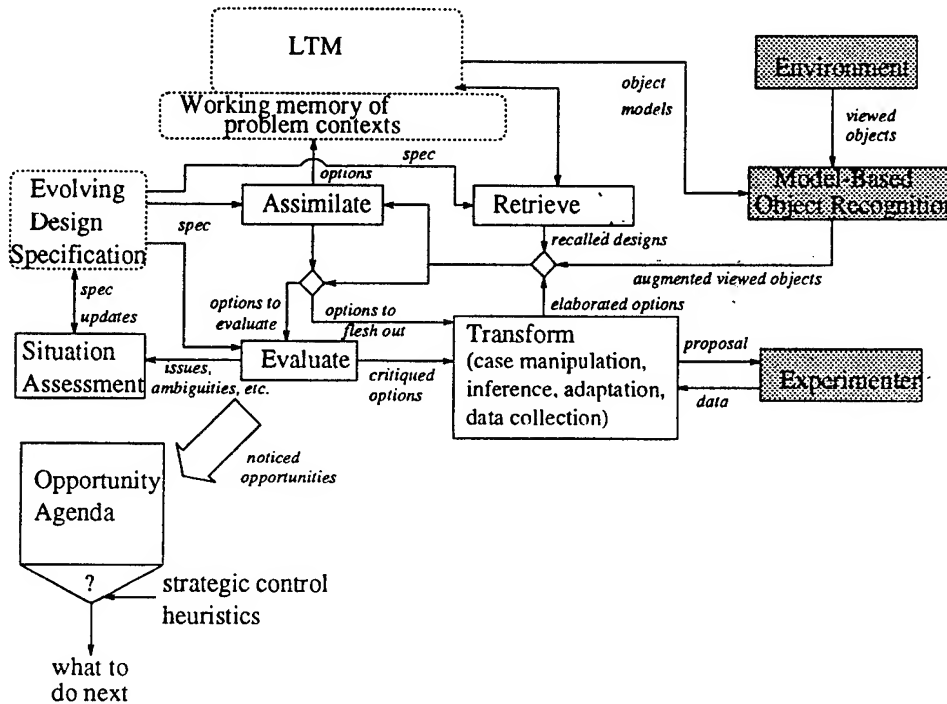


Figure 2. Architecture of Improviser.

We are developing an experimental system to computationally model case-based creative design, concentrating in particular on the processes of preparation, assimilation, and control of focus underlying perceptive observation. We are implementing this model in a system called IMPROVISER (Invention Modeled by Problem Redescription, observation, and evaluation, Interacting SERendipitously). Its architecture (shown in Figure 2) has the following primary components:

1. **Problem evolution**, which is key to preparation, is modeled using situation assessment procedures co-routined with evaluation techniques. For example, as an option is evaluated in accordance with the current problem specification, there might be a constraint that is not refined or operationalized enough for the evaluation process to check it. This would reveal an opportunity for the problem constraint to be refined in the context of a particular design option. Also as data is collected from experiments and desirable or undesirable features are noticed, new constraints are added to require, prefer, prohibit, or

avoid the observed feature. This involves performing blame assignment by tracing the causes and choosing what to constrain that will fix the problem. IMPROVISER uses knowledge about what can be constrained and what types of constraints can be possibly satisfied. It also uses knowledge about inverses of features in order to generate constraints to prohibit undesirable behavior or properties.

2. **Solution transformation** is a set of mechanisms that modify proposed solutions, or "options", (which are either previous design cases that have been recalled, objects that are viewed in external environment, or previously adapted options). These typically elaborate the option by merging it with other options, by augmenting it with experimental data or results of simulation or case-based projection, or by applying case adaptation strategies to it. They may also remove features from the option or create an approximation to it in the form of an experimental prototype that can be tested in limited, controlled fashion.
3. The **memory** consists of a *long-term* component, a *working memory* component, and a *focal store*. The long-term memory contains a library of previous design cases. The working memory component holds the design options under consideration for the design problem and its various subproblems. These are held in structures called *problem contexts*, which are organized according to the current structure of the problem framework. The design of the problem contexts, including how they interrelate, is a key part of our model of assimilation and preparation processes and is described in more detail in Section 5.1. The focal store contains information about the current state of the reasoner, including the subproblem being considered, the task at hand, the options currently in focus.

The two main operational components that work on the content of the memory are *memory retrieval* and *update* processes. The evolving specification is used as a probe to recall relevant design cases (for evaluation, elaboration, etc.). It also informs memory update. It is the main source of information about the current problem framework and what criteria and constraints are relevant. Memory update accumulates design options proposed (i.e., those retrieved, elaborated, or viewed directly in the external environment) into problem contexts. Within the problem contexts, design options are organized and compared with respect to each other, along the dimensions relevant to the

problem specification. This is described more fully in Section 5.1.

4. The control of IMPROVISER is quite flexible in order to facilitate the kinds of opportunism inherent in creative reasoning and allow the necessary interleaving and communication among the component processes. Rather than a rigid control structure IMPROVISER has a blackboard-style architecture, guided by explicit strategic control heuristics. In particular, a set of monitoring procedures, called "noticers," are associated with each process which watch for opportunities for a specific type of processing to occur. The opportunities noticed are placed on an "opportunity" agenda, maintained and accessed by strategic control heuristics.

For example, a noticer associated with the assimilation process watches for an alternative to be added that is much better than any other alternative proposed so far, along some desired criterion. This yields an opportunity to change the problem description by increasing the priority of that criterion and/or by relaxing constraints that are not met by that proposal. This simulates the behavior of changing the relative importance among criteria to accommodate an unexpectedly good solution that is stumbled upon. An example strategic control heuristic would be to pursue elaboration opportunities for alternatives that satisfy a desired criteria extremely well before pursuing evaluative processes that would negatively critique the alternatives. This simulates the behavior of optimistically pursuing an idea, suspending all but constructive criticism.

5. Obtaining basic perceptual information about the environment, the running of experimental prototypes and providing feedback about their results is all done manually, through an oracle that provides canned responses to requests for experimental or environmental information. In other words, the oracle takes the place of an external agent (e.g., a human or robot) that physically interacts with the external environment to perform such activities as constructing actual prototypes, running them, and making observations. (External agents that are simulated by the oracle are shown as shaded boxes in Figure 2, since building them is outside the scope of this research.) This allows us to investigate in a highly controlled fashion what is the content and focus of a designer's interactions with the surrounding, physical environment.

5.1. PROBLEM CONTEXTS

In solving complex design problems, designers break the problem into parts and generate ideas for each. The subproblems might be related by dependencies between and sharing of constraints and by interactions among the variables of the design. A simple example is the cost constraint in the ME design which restricted the total cost of the device to be less than one hundred dollars. This resource constraint is shared across all subproblems and the choice of a certain solution to one subproblem, has an effect on how much of the resource can be expended in solving other subproblems. Likewise, the choice of spring strength, which is a variable in the spring launch device, has an effect on the launch force which effects the ultimate speed during transport and the amount of cushioning material needed in solving the passenger protection subproblem.

We capture the structure of the design problem, including its component subproblems and their interrelationships, in a network of structures called *problem contexts*. The content, structure, and interrelationships of problem contexts play an important role in our models of the assimilation and preparation processes.

Each problem context represents a subproblem of the overall design problem. It contains the constraints of the subproblem, the ideas that have been proposed so far for solving the subproblem, and information about how well these ideas satisfy the constraints.

Problem contexts are organized along two orthogonal dimensions, which can be roughly characterized as "vertical" and "horizontal." Vertical relationships represent either *component* relationships (i.e., that one problem is a subproblem or part of another) or *conditional refinement* relationships. Refinement relationships represent an extension or specialization of the problem that results from making some design decision. For example, if a decision is made to have a powerful, sudden launch, the constraints on amount and type of cushioning needed become more concrete. A child problem context resulting from a refinement has additional and/or more specialized constraints than the parent problem context. It also inherits all the constraints of the parent that are consistent with the decision made. The component relationships capture the hierarchical decomposition of the problem, while the refinement relationships capture the successive refinement of the problem.

In addition to these vertical relationships, there are horizontal ones that represent interrelationships between the component subproblems of a problem. These

are encoded as relationships between design variables and the constraints on them. A key role these relationships play is in explaining and modeling the control of focus during design, experimentation (or testing), and interpretation. This is described in Section 5.2.

Each problem context contains these pieces of information:

1. One is a set of **constraints** that a solution to the problem must satisfy. Each constraint specifies a *design variable* that is being constrained and which concrete *features* of a solution it corresponds to (e.g., "launching spring strength" is a design variable, while Hook's constant ("k") of a given spring is a concrete, measurable feature of the solution object). In addition, the constraint specifies a range of *desired values* and a range of *forbidden values* for the design variable. These values need not be numeric. They may be specified by a general unary predicate that is used to determine whether the constraint is satisfied.

There are two other important pieces of knowledge associated with constraints. One is a binary *comparator* predicate which decides which of two values better satisfies the constraint. This is needed in doing relative comparisons among design options. The other piece of knowledge is a unary function for determining "*direction of improvement*." It determines whether a given feature value is close to or better or worse than the ideal value satisfying the constraint. This is important in specifying which end of a range of satisficing values is preferred. This is needed to compute degree of match and to judge whether expectation violations are better or worse than what was expected.

2. The problem context also contains a set of **options** which are the solution ideas considered so far for satisfying the problem's constraints. These may have come from a long-term memory of design cases, from externally viewed objects, or from adaptations of other options previously considered. Each option contains case-specific knowledge about the alternative, such as how it has succeeded or failed in the past, what evaluation criteria it satisfied well or failed to satisfy in the past, and consequences of using it in specific situations. It will also have any model-based knowledge about the alternative that has been learned in the past. For example, in our bending springs example, once the toilet paper holder, viewed in the external environment, was recognized, a model of its typical behavioral and functional properties was recalled. This model and specific experiences (cases) are part of the representation of the

paper holder.

3. Finally, **annotated relationships** bidirectionally link each constraint with each option explored so far. The annotations rate how well the option satisfies the constraint. This graph structure makes it easy to compare options along a particular problem dimension as well as facilitating the evaluation of a particular option over all the constraints that bear on it.
4. Each problem context also contains a **priority structure** specifying the current relative importance among the constraints. This is given in the form of a partial ordering of constraints.

Problem contexts may overlap in that they may share constraints and/or options. Our representation of problem contexts is similar to the *question-option-constraint* (or QOC) models used in capturing design decisions and rationale in user-interface design (Maclean, *et al.*, 1991).

Problem contexts are dynamically created and restructured with respect to each other as the problem specification evolves and as proposed solution options are explored. They are central to modeling preparation and assimilation. As the problem specification evolves, the focus changes on the relevant descriptors (the focus of constraints) to be used for organizing options in memory (e.g., shape, construction cost, personal safety). When an option is entered into memory, it is interpreted with respect to the descriptors in the subproblem contexts to find the best place(s) to store the option.

Serendipitously stumbling across a solution and recognizing that it is relevant to a pending problem is the result of storing the option in a problem context, in relation to other previously collected options, and noticing that the new option is a much closer match to the desired solution than anything previously considered (Wills and Kolodner, 1994a). The structure of the problem context makes it easy to notice when a constraint is satisfied extremely well or poorly, relative to what's been explored so far.

The various subproblem contexts can be seen as dynamically constructed models of desired solutions, built during problem evolution. Recognizing an option as a solution results when an alternative is stored that is a relatively close match to the desired solution model.

The management of problem contexts in working memory is an interesting area of active research (Simina and Kolodner, 1995). The issues being investigated include: how are problems suspended when an impasse or interruption occurs;

how are they recalled when something relevant to solving them is found; how do they change as other related problems are addressed; when are suspended problems forgotten? Recently, (Simina and Kolodner, 1995) proposed a new model of working memory that augments previous models of opportunistic reasoning, combining predictive encoding ideas (Patalano, *et al.*, 1993; Hammond, 1989) with active monitoring mechanisms (Birnbaum and Collins, 1984; Birnbaum, 1986).

5.2. DESIGN VARIABLE DEPENDENCY GRAPH

While working on a complex design problem, designers move fluidly between the various subproblems, often seeing the relevance of some design option to more than one problem at once. As decisions are made, their consequences are propagated to other subproblems. The way that one subproblem is reformulated or redescribed can have a profound influence on how another subproblem is viewed.

IMPROVISER takes a step toward understanding and explaining these shifts in focus across subproblems as well as shifts in the focus of attention during experimentation and interpretation of results. It does this using a representation of relationships among design variables and the constraints on them, in the form of a *design variable dependency graph*.

In this graph, the nodes each represent a design variable or parameter whose value the designer needs to decide. Example design variables in the ME design project were the "passenger capacity" (i.e., how many eggs would the device carry) and "striking distance" (i.e., how far would the device travel from its starting location).

Edges in the design variable dependency graph represent information about which variables influence which others. The variable represented by the source of the edge *influences* the variable represented by its sink. Edges are annotated with information about the nature of the influence – *direct* or *inverse*. If x directly influences y , then a change in the value of x will cause the same direction of change to occur in y (e.g., if both are values, an increase in x will cause y to increase). Inverse influences, on the other hand, cause changes in opposite directions (e.g., if x and y are values, an increase in x will cause y to decrease).

Each design variable has a set of constraints on it, as described in the previous section. Multiple, possibly conflicting constraints often are imposed on a single

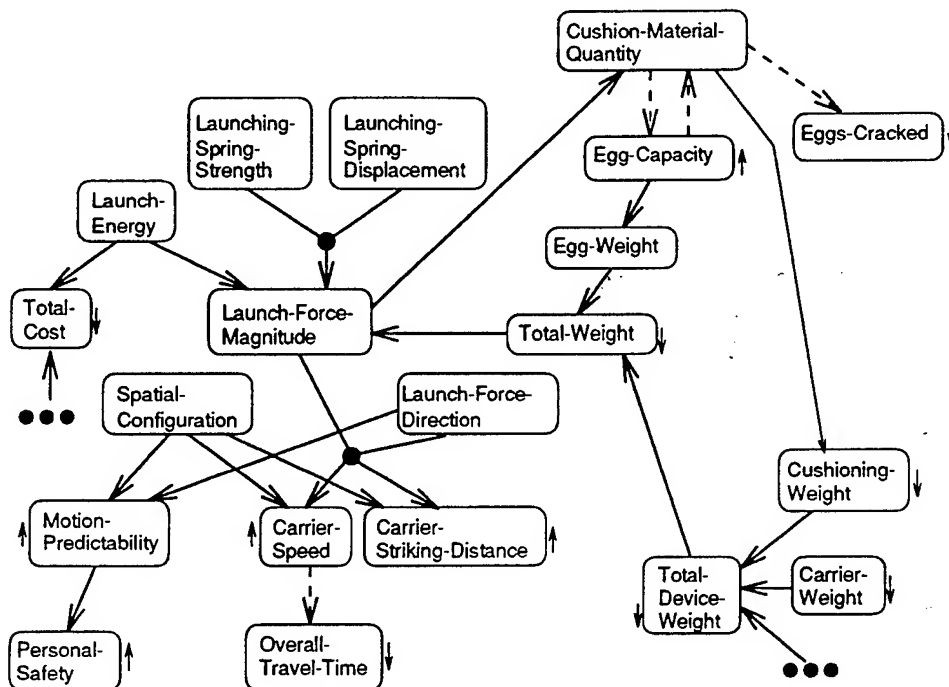


Figure 3. A design variable dependency graph. Nodes represent design variables. Solid edges represent direct influences; dashed edges represent inverse influences. Solid circles merging multiple edges represent aggregate design variable influences. Small arrows adjacent to nodes illustrate current direction of improvement constraints on design variables that represent quantities.

design variable. Figure 3 shows a portion of the design variable dependency graph capturing variable relationships at one point in the ME design project.

IMPROVISER derives relationships between constraints based on the underlying design variable dependency graph and based on the *direction of improvement* the constraint specifies. (These are then cached in the representation of the constraints as the relationships are revealed.) The two primary constraint relationships that are derived are **trade-off** and **concur** relationships.

Two constraints C_1 and C_2 **trade-off** iff either

1. there is a *direct influence* relationship between the design variables constrained by C_1 and C_2 , but an *opposite* direction of improvement sought by the two constraints; or
2. there is an *inverse influence* relationship between the design variables constrained by C_1 and C_2 , with the *same* direction of improvement sought by

the two constraints.

Conversely, two constraints C_1 and C_2 **concur** iff either

1. there is a *direct influence* relationship between the design variables constrained by C_1 and C_2 , with the *same* direction of improvement sought by the two constraints; or
2. there is an *inverse influence* relationship between the design variables constrained by C_1 and C_2 , but an *opposite* direction of improvement sought by the two constraints.

For example, there is a trade-off relationship between the energy required and the device speed in Figure 3. (Direct influence relationships are transitive.)

Design variables may be *aggregates* in that they are a set of design variables that together influence other variables. The consequences of changing one of them cannot be derived or judged independent of knowledge of changes in the other. In the example graph shown in Figure 3, launching-spring-strength and launching-spring-displacement are aggregates, as are launch-force-magnitude and launch-force-direction. When one of these variables is in focus, so are those with which it is aggregated.

The knowledge captured in the design variable dependency graph is useful for:

- detecting trade-offs in constraints on the variables being reasoned about at any given point in the design process;
- propagating consequences of design decisions;
- capturing connections between problem contexts;
- judging whether an expectation violation is good or bad (using the direction of improvement for the constraints involved);
- deriving consequences of expectation violation (by following influences relations); and
- defining the space of features that are relevant in a design problem and explaining attentional focus shifts as movement in this space.

These features are best explained in the context of a demonstration scenario, which is provided in the next section.

6. Scenario

To show how IMPROVISER uses the computational mechanisms described above to model the processes underlying observation, we present a demonstration scenario in this section. The focus of the scenario is the bending springs example described in the introduction. The key issues to keep in mind while considering this scenario are the following.

- What observations are made at each point and how is focus directed toward them? That is, what is noticed and why is that noticed over other possibilities? This includes focus on specific pieces of experimental data, focus on particular consequences of what is observed, as well as focus on anomalies, invariants, and extremes.
- Which mechanisms prepare the reasoner to interpret observations in the way it does? For example, how does it learn what features to pay attention to, what organizational structure and evaluative dimensions to use to compare, contrast, and organize ideas?
- How are departures from the current organizational structure expectation violations detected? How are their consequences derived? How is their significance and relevance to the design judged?
- Once “interesting” observations are made, how does the reasoner incorporate them into the design, for example, by refining, elaborating, or reformulating the problem specification?

The IMPROVISER scenario highlights these issues and provides initial answers and explanations.

IMPROVISER starts with a partial specification which includes specifications for each subproblem in the current partitioning of the problem (launching, moving, stopping, and protecting the eggs). The launch subproblem specification contains a partial specification for a spring launch mechanism. (In the following, “??” denotes incompleteness due to pending decisions; “...” denotes parts of the specification not shown. IMPROVISER’s output is shown in **bold face**, with typewriter font showing the data it is manipulating.)

```
<Spec:
Subproblem: Launch
  Parts: Spring, Base
  Attached(Spring, Base, <position>)
  Spring:
```

```

k: ?? ;; force constant
x: ?? ;; spring displacement
Launch-Force: (- (* k x)) ;; Hooke's law
...
Subproblem: Protect-Eggs ...
Subproblem: Transport ...
Subproblem: Stop ...
...>

```

This specifies that the launch mechanism should consist of two parts, attached to each other in a particular configuration (given in <position>). There is a pending decision as to the choice of spring strength (k) and how much it should be compressed (x) to achieve a certain launch force. (There are several other constraints involving the launch force which are not shown, such as constraints relating striking distance and launch force or relating the type and amount of egg protection material with the launch force it must cushion.)

Note that the expected behavior is incomplete; there are no constraints that ensure that the spring stays straight during compression (and release). Our designers did not think of the possibility that the spring would bend. They didn't even worry about gravity affecting its horizontal position. They had a misconception (or simplifying assumption) that the springs were laterally rigid. The pictures they drew didn't show any required support around springs in any of the spring launch mechanisms. The support requirement emerged later.

Bringing a new design variable into focus:

LAUNCHING-SPRING-STRENGTH (K)

IMPROVISER chooses to work on the pending decision concerning spring strength and to employ a experimental (trial-and-error) method to help make the decision (i.e., examine the springs that are available and compare their strengths to form a more refined constraint). These choices are based on the knowledge IMPROVISER has about the design variables involved in the pending decisions, such as whether sensory feedback is useful in collecting data about which range of values would be applicable and whether the ultimate choice of values is dependent on what values are exist in available design pieces. These pieces of knowledge are contained in attributes on the design variables (e.g., sensory-feedback-useful and availability-dependent are attributes). IMPROVISER also bases these choices on information about its current environment (e.g., the current location, a hard-

ware store, is a place where design pieces are available with a range of relevant parameters).

The standard way to refine a constraint using experimentation is to construct a set of nearly identical prototypes each of which uses a different value for the design variable whose value needs to be constrained. Each prototype consists of enough of the design to measure and compare design variable values. Automatically constructing these prototypes is an intriguing open problem which is beyond the scope of this research (see Section 7). For now, IMPROVISER asks the oracle to create the prototypes, run them, and report the results.

Nature of Oracle:

(PHYSICAL-MANIPULATION SENSORY-OBSERVATION)

Oracle has been asked to perform a[n] EXPERIMENT.

The object is: <OPTION FINGER-SPRING-PROTOTYPE>.

It has been instructed to:

1. "squeeze spring by moving thumb toward finger;"
2. "allow spring to push apart the fingers;"
3. "measure force pushing fingers apart."

In addition to LAUNCHING-SPRING-STRENGTH (K), these variables come into focus:

Bringing new design variables into focus:

LAUNCHING-SPRING-DISPLACEMENT (X)

LAUNCH-FORCE-MAGNITUDE (FORCE-MAGNITUDE)

LAUNCH-FORCE-DIRECTION (FORCE-DIRECTION)

The design variable dependency graph is used to determine this. LAUNCHING-SPRING-DISPLACEMENT (X) is aggregated with LAUNCHING-SPRING-STRENGTH (K), so it is automatically brought into focus, since the two act together. LAUNCH-FORCE-MAGNITUDE is directly influenced by spring displacement and strength, so it is brought into focus. LAUNCH-FORCE-DIRECTION is also brought in because it is aggregated with magnitude.

This raises the issue: how much of what is influenced by spring strength gets pulled into focus? Blindly following influence relations will draw in nearly the entire variable dependency graph. What limits the focus is the experimentation task. IMPROVISER will only focus on those variables that are relevant to the experimental prototype. For example, striking-distance and amount of egg cushioning are not relevant.

Part of the information the oracle provides IMPROVISER is the mapping from features and variables in the design to those in the experimental prototype. This

information is crucial in transferring and interpreting results from the experimental world to the design world. Given this mapping, IMPROVISER tells the Oracle what experimental prototype features are currently in focus.

Oracle should focus on the features:

(FORCE-DIRECTED-AT-FINGERS FORCE-SEPARATING-FINGERS X K)

The oracle compares several springs and feeds back experimental data to IMPROVISER. The data is augmented with causal information about how the data resulted from the properties of the partial design. The oracle reports that the spring bends, which causes it to exert a weak force in the direction of its axis.

Oracle's observations:

((BENT-SPRING SPRING-SHAPE BENT)

(FORCE-MAGNITUDE FORCE-SEPARATING-FINGERS 23)

(FORCE-DIRECTION FORCE-DIRECTED-AT-FINGERS 45))

Causes:

BENT-SPRING -> FORCE-MAGNITUDE

BENT-SPRING -> FORCE-DIRECTION

IMPROVISER notices that these results are not what is expected. How does it do this? First of all, the expectations come from the constraints in the current specification. IMPROVISER notices that they are violated as it transfers data from the experiment back to the design and uses the data to elaborate the spring launch option. Note that only the data related to features currently in focus are transferred. Also, evaluation of the option with respect to specification is a by-product of this elaboration and assimilation process.

Transferring experimental datum since feature is in focus:

(FORCE-MAGNITUDE FORCE-SEPARATING-FINGERS 23)

Transferring experimental datum since feature is in focus:

(FORCE-DIRECTION FORCE-DIRECTED-AT-FINGERS 45)

<OPTION FINGER-SPRING-PROTOTYPE> violates a SPECIFICATION constraint:

Feature FORCE-DIRECTED-AT-FINGERS has NONMATCHING value 45.

<OPTION FINGER-SPRING-PROTOTYPE> violates a SPECIFICATION constraint:

Feature FORCE-SEPARATING-FINGERS has NONMATCHING value 23.

As the violations are detected, their significance is judged. This is based on how extreme the value is, compared to those of previously considered options (if any) and it's based on how different the actual value is from the expected one, which is determined using the constraint's binary comparator predicate and direction of improvement function. Differences from expectations are labeled as high, low, or uncertain and as having a positive or negative direction.

Significant differences detected:

(<DIFF FORCE-DIRECTED-AT-FINGERS -H>
 <DIFF FORCE-SEPARATING-FINGERS -H>)

Consequences of significant expectation violations are derived and used to further elaborate the design option. These are of the form of general characterizations of whether a particular design variable is influenced in a negative or positive way, based on how it is related to the variable involved in the expectation. For example, the force along the spring's axis is weaker than the ideal launch force ($F = -kx$), which will make the device move slower and not as far as desired.

Derived consequences of experimental results on:

<OPTION FINGER-SPRING-PROTOTYPE>:

As consequence of

<DIFF FORCE-DIRECTED-AT-FINGERS -H>,

noticing NEGATIVE impact on:

MOTION-PREDICTABILITY (PREDICTABILITY)

CARRIER-SPEED (SPEED)

CARRIER-STRIKING-DISTANCE (STRIKING-DISTANCE)

As consequence of

<DIFF FORCE-SEPARATING-FINGERS -H>,

noticing NEGATIVE impact on:

CARRIER-SPEED (SPEED)

CARRIER-STRIKING-DISTANCE (STRIKING-DISTANCE)

In addition, IMPROVISER traces the causes of unexpected features and updates the specification to prohibit (or require) them. To do this, IMPROVISER uses the causal information provided by the oracle. However, in general, discovering the constraints to add to the specification which will require or prohibit some observed feature of a device involves reasoning based on a causal model of the device (Bhatta, *et al.*, 1994). IMPROVISER also makes use of knowledge about what can be constrained and what types of constraints can be possibly satisfied. It relies on knowledge about inverses of features in order to generate constraints to prohibit undesirable behavior or properties.

Adding a newly emerged constraint:

<C: LAUNCHING-SPRING-SHAPE=(STRAIGHT)> to <SPRING-4674>

In addition to revealing new constraints, this process helps draw attention to existing constraints that need to be refined or satisfied.

Constraint to try harder to satisfy:

<C: LAUNCH-FORCE-DIRECTION=(ORTHOGONAL-TO-LAUNCHED-OBJ?)>

Given a set of constraints that are currently in focus, IMPROVISER chooses one to pursue, using strategic control heuristics. These base the choice on temporal information (e.g., what major changes have been made to the problem specification? which constraints have newly emerged? which constraints have most recently been pursued?), availability of methods for refining or satisfying the constraint, the dependencies among the constraints, and constraint priorities. In this case, IMPROVISER chooses to try to satisfy the newly emerged constraint on spring shape, partly because it is a new constraint and partly because of constraint dependencies: the difference in shape caused the other observed data.

Opportunity chosen:

<OPP: SATISFY-CONSTRAINT

(<C: LAUNCHING-SPRING-SHAPE=(STRAIGHT)>)>

Bringing a new design variable into focus:

LAUNCHING-SPRING-SHAPE (SPRING-SHAPE)

DVars(Features) currently in focus:

LAUNCHING-SPRING-SHAPE (SPRING-SHAPE)

LAUNCH-FORCE-DIRECTION (FORCE-DIRECTION)

LAUNCH-FORCE-MAGNITUDE (FORCE)

LAUNCHING-SPRING-DISPLACEMENT (X)

LAUNCHING-SPRING-STRENGTH (K)

A standard method of forcing a small object to maintain a desired shape: holding it in that shape with your hand. IMPROVISER asks the oracle to do this and determine whether the spring stays straight.

Nature of Oracle:

(PHYSICAL-MANIPULATION SENSORY-OBSERVATION)

Oracle has been asked to perform a[n] EXPERIMENT.

The object is <OPTION WRAPPED-HAND-SPRING-PROTOTYPE>.

It has been instructed to:

1. "wrap one hand around spring;"
2. "with other hand, squeeze spring by moving thumb toward finger;"
3. "hold spring uniformly snug;"
4. "determine whether spring stays straight."

Oracle should focus on the features:

(SPRING-SHAPE FORCE-DIRECTED-AT-FINGERS FORCE-SEPARATING-FINGERS X K)

This requires extending <LAUNCH-4673> **design w/ a subspec for additional structure:** <SPRING-SUPPORT-4720>.

The results are that the spring stays straight, but its compression is hindered by the wrapped hand.

Oracle's observations:

```
((STRAIGHT-SPRING SPRING-SHAPE STRAIGHT)
 (FORCE-MAGNITUDE FORCE-SEPARATING-FINGERS 10)
 (FORCE-DIRECTION FORCE-DIRECTED-AT-FINGERS 0)
 (LIMITED-DISPLACEMENT X 5)
 (LATERAL-RIGIDITY BEHAVIOR RADIUS-CONSTANT)
 (LONGITUDINAL-RIGIDITY BEHAVIOR LENGTH-CONSTANT)
 (SNUG-FIT-R RADIUS 25.1)
 (SNUG-FIT-L LENGTH 150)
 (SNUG-FIT-SHAPE SHAPE CYLINDRICAL))
```

Causes:

```
STRAIGHT-SPRING -> FORCE-DIRECTION
LIMITED-DISPLACEMENT -> FORCE-MAGNITUDE
LATERAL-RIGIDITY -> STRAIGHT-SPRING
SNUG-FIT-R -> STRAIGHT-SPRING
SNUG-FIT-L -> STRAIGHT-SPRING
SNUG-FIT-SHAPE -> STRAIGHT-SPRING
LONGITUDINAL-RIGIDITY -> LIMITED-DISPLACEMENT
```

Again, the relevant experimental data is transferred and assimilated. IMPROVISER notices both positive and negative unexpected results. The spring shape is now straight, but the spring displacement is undesirably limited. This will affect device speed and striking distance.

Transferring experimental datum since feature is in focus:

```
((STRAIGHT-SPRING SPRING-SHAPE STRAIGHT)
 (FORCE-MAGNITUDE FORCE-SEPARATING-FINGERS 10)
 (FORCE-DIRECTION FORCE-DIRECTED-AT-FINGERS 0)
 (LIMITED-DISPLACEMENT X 5))
```

Datum (LIMITED-DISPLACEMENT X 5) VIOLATES

```
<C: LAUNCHING-SPRING-DISPLACEMENT=(MAX-SPRING-COMPRESSION?)>
WORSE than expected ((100)).
```

```
<OPTION WRAPPED-HAND-SPRING-PROTOTYPE> satisfies a
SPECIFICATION constraint:
```

```
Feature FORCE-DIRECTED-AT-FINGERS has value 0, rated E.
```

```
<OPTION WRAPPED-HAND-SPRING-PROTOTYPE> violates a
SPECIFICATION constraint:
```

```
Feature FORCE-SEPARATING-FINGERS has NONMATCHING value 10.
```

```
<OPTION WRAPPED-HAND-SPRING-PROTOTYPE> satisfies a
SPECIFICATION constraint:
```

```
Feature SPRING-SHAPE has value STRAIGHT, rated E.
```

Significant differences detected:

```
(<DIFF X -H>
<DIFF FORCE-DIRECTED-AT-FINGERS +H>
<DIFF FORCE-SEPARATING-FINGERS -H>
```

<DIFF SPRING-SHAPE +H>

Derived consequences of experimental results on:

<OPTION WRAPPED-HAND-SPRING-PROTOTYPE>:

As consequence of

<DIFF X -H>,

noticing NEGATIVE impact on:

LAUNCH-FORCE-MAGNITUDE (FORCE)

As consequence of

<DIFF FORCE-DIRECTED-AT-FINGERS +H>,

noticing POSITIVE impact on:

MOTION-PREDICTABILITY (PREDICTABILITY)

CARRIER-SPEED (SPEED)

CARRIER-STRIKING-DISTANCE (STRIKING-DISTANCE)

As consequence of

<DIFF FORCE-SEPARATING-FINGERS -H>,

noticing NEGATIVE impact on:

CARRIER-SPEED (SPEED)

CARRIER-STRIKING-DISTANCE (STRIKING-DISTANCE)

IMPROVISER adds new constraints to borrow features it views as positive (e.g., cylindrical shape) and to prohibit undesirable features (e.g., constant length). It reasons about the causes of the spring compression hindrance and updates the specification to require the tube to be collapsible (i.e., to allow its length to vary).

Adding a newly emerged constraint:

<C: L-BEHAVIOR=(LENGTH-VARIES)> to <SPRING-SUPPORT-4720>

Constraint to try harder to satisfy:

<C: LAUNCHING-SPRING-DISPLACEMENT=(MAX-SPRING-COMPRESSION?)>

Adding a newly emerged constraint:

<C: SPRING-SUPPORT-SHAPE=(CYLINDRICAL)> to <SPRING-SUPPORT-4720>

Adding a newly emerged constraint:

<C: SPRING-SUPPORT-RADIUS=(25.1)> to <SPRING-SUPPORT-4720>

Adding a newly emerged constraint:

<C: R-BEHAVIOR=(RADIUS-CONSTANT)> to <SPRING-SUPPORT-4720>

Constraint to try harder to satisfy:

<C: LAUNCHING-SPRING-DISPLACEMENT=(MAX-SPRING-COMPRESSION?)>

Constraint to try harder to satisfy:

<C: LAUNCH-FORCE-MAGNITUDE=(LARGE-FORCE?)>

Not only did several constraints newly emerge, but an entire new subspecification for the subproblem of supporting the spring was added. The opportunity to generate options for this specification is chosen next.

Heuristic used to choose next opportunity:

#<Compiled function CHOOSE-EXPLORE-NEW-SPEC 21010133733>

DTIC COULD NOT GET MISSING
PAGE FROM CONTRIBUTOR

Opportunity chosen:

<OPP: RECALL-OPTIONS
 (<SPRING-SUPPORT-4720>)>

Constraints:

<C: R-BEHAVIOR=(RADIUS-CONSTANT)>
 <C: SPRING-SUPPORT-RADIUS=(25.1)>
 <C: SPRING-SUPPORT-LENGTH=(150)>
 <C: SPRING-SUPPORT-SHAPE=(CYLINDRICAL)>
 <C: L-BEHAVIOR=(LENGTH-VARIES)>

Using this specification as a probe to memory, IMPROVISER tries to recall a device that will satisfy it, but the retrieval comes up empty.

Trying to recall options for current specification.

Retrieval results: NIL

Since no viable options are found, IMPROVISER suspends work on the launch subproblem and switches to a different problem context for the pending problem of how to protect eggs. How this other problem is chosen and how suspended problems are managed and resumed when an opportunity exists to solve them is an interesting area of active research. (It is not the subject of this paper, but see (Wills and Kolodner, 1994a; Simina and Kolodner, 1995).)

Switch problem context to egg protection.

<Spec <CUSHIONING-MATERIAL-2984>
 ...
 Subproblem: Protect-Eggs
 Parts: Cushioning-Material
 Enclosed(Egg, Cushioning-Material)
 Cushioning-Material:
 Cost: cheap
 Pressure-Resistance: Soft
 Weight: Light
 ...>

While looking for objects that satisfied this description, the oracle reports that a toilet paper holder is observed. The observation is a mix of image features and knowledge about the holder, once it has been identified through object recognition. The most relevant, recent pending problem context is retrieved and the paper holder is assimilated into it, based on its structural, imagistic, and behavioral properties.

ORACLE: An object is viewed and recognized as a TPH:

OBSERVED-TPH

Assimilating object into spec: <SPRING-SUPPORT-4720>

and managed in working memory. Some of these issues were mentioned at the end of Section 5.1. Additional issues include: What influences which problem contexts in working memory are active (e.g., recency, interaction between the problems)? How do they change as related problems are worked on? How do they decay? How does knowledge of functional properties of an object inhibit the retrieval of a relevant problem context in which the object can be used in a new way. (This is important in modeling functional fixedness (Duncker, 1945; Maier, 1931; Maier, 1970).)

Another future direction concerns how we model opportunistic control and its relation to working memory. We currently maintain an agenda of pending opportunities for action, which keeps track of possible next steps IMPROVISER can take. This "opportunity agenda" is currently separate from the working memory. However, its access and maintenance are tightly coupled with the structure of the problem contexts that are in focus or that are pending. In fact, we have noticed that there is a striking similarity between the heuristics and processes used in recognizing an opportunity to resume a suspended problem and those used to detect goal satisfaction opportunities in general. We intend to merge the opportunity agenda into the working memory for a unifying treatment of both units of control and units of problem representation.

In addition, our study of ME designers has pointed out the importance of real-world experimentation as a source of both evaluative feedback and solution ideas. In the process of constructing experimental, concrete prototypes and trying them out, new constraints on the problem and new problems to solve arose. Being able to do this experimentation requires being able to construct prototypes, each an approximation that focuses on some subset of functionality. (For example, our ME designers constructed cushioning devices for single eggs to try out various materials and become familiar with their properties.) But how people do this is not yet well understood.

Prototype construction seems related to the problem of setting up thought experiments in scientific discovery (Nersessian, 1993). In fact, the process of isolating portions of behavior or functionality into an approximation or prototype of the design seems to have a great deal in common with the *constructive modeling* process (Clement, 1989; Nersessian, 1992; Nersessian, 1993), being studied in the context of scientific discovery. In constructive modeling, new hybrid analogical models are generated by merging pieces of knowledge from multiple source

domains, through a combined process of analogical reasoning, visual reasoning, and thought experimentation. Exploring the connections between prototype construction and constructive modeling may shed light on both.

Another interesting aspect of experimentation is that solutions often evolve during the experiment as problems are found and the prototype is "patched" or quickly modified to push it through the exercise. Sometimes this involves physical actions that respond to affordances in the experimental situation. These are typically primitive, physical actions similar to those observed by (Kohler, 1925). For example, we observed this in the bending springs example when the ME designer wrapped a hand around the spring to keep it from bending. This suggested the partial solution of enclosing the spring in a tube. It would be interesting to investigate how research on affordances bears on idea generation in general.

References

- Bhatta, S., Goel, A., and Prabhakar, S.: 1994. Innovation in Analogical Design: A Model-Based Approach. In the *Proceedings of the Third International Conference on Artificial Intelligence in Design (AID-94)*, Lausanne, Switzerland, pp. 57-74.
- Birnbaum, L.: 1986. Integrated Processing in Planning and Understanding. Ph.D. thesis, Yale.
- Birnbaum, L. and Collins, G.: 1984. Opportunistic Planning and Freudian Slips. In *Proc. of the Sixth Conference of the Cognitive Science Society*.
- Clement, J.: 1989. Learning via Model Construction and Criticism. In Glover, G., Ronning, R., and Reynolds, C. (eds.), *Handbook of Creativity: Assessment, Theory and Research*, pp. 341-381. New York: Plenum.
- Corrigan, R. W.: 1979. *The World of the Theatre*. Glenview, IL: Scott, Foresman and Co.
- Domeshek, E. A., and Kolodner, J. L.: 1993. Using the points of large cases, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 7(2): 87-96.
- Duncker, K.: 1945. On problem solving. *Psychological Monographs*, 58(270).
- Fischer, G.: 1993. Turning Breakdowns into Opportunities for Creativity. In *Proceedings of the International Symposium on Creativity and Cognition*, Loughborough, England.
- Gero, J.: 1990. Design Prototypes: A Knowledge Representation Schema for Design, *AI Magazine*, 11(4), Winter.
- Gero, J. and Maher, M.: 1993. *Modeling Creativity and Knowledge-Based Creative Design*. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.
- Goel, V. and Pirolli, P.: 1989. Motivating the Notion of Generic Design within Information Processing Theory: The Design Problem Space. *AI Magazine*, Vol. 10, No. 1, pp. 18-36.
- Hammond, K.: 1989. Opportunistic memory. In *IJCAI-89* (pp. 504-510). Detroit, MI.
- Jones, E. K.: 1992. The Flexible Use of Abstract Knowledge in Planning. Northwestern University, Institute for the Learning Sciences Technical Report no. 28.
- Kaplan, C. and Simon, H.: 1990. In Search of Insight. *Cognitive Psychology*, Vol. 22, pp. 374-419.
- Kohler, W.: 1925. *The Mentality of Apes*. New York: Liveright.

- Kolodner, J.: 1983. Reconstructive Memory: A Computer Model. *Cognitive Science* 7(4): 281-328.
- Kolodner, J. L.: 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan-Kaufmann.
- Kolodner, J.L.: 1994. Understanding Creativity: A Case-Based Approach. In S. Wess, K.D. Althoff and M.M. Richter (eds.), *Topics in Case-Based Reasoning*, selected papers from the *First European Workshop on Case-Based Reasoning*. Kaiserslautern, Germany. Springer-Verlag. Lecture Notes in Artificial Intelligence.
- Kolodner, J. L. and Penberthy, T. L.: 1990. A Case-Based Approach to Creativity in Problem Solving. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. August.
- Kolodner, J. L. and Wills, L. M.: 1993. Case-Based Creative Design. *AAAI Spring Symposium on AI and Creativity*. Stanford, CA. March 1993. Reprinted in *AISB Quarterly* special issue (no. 85) on AI and Creativity, edited by Terry Dartnall, Autumn 1993.
- Kolodner, J. L. and Wills, L. M.: 1993. Paying Attention to the Right Thing: Issues of Focus in Case-Based Creative Design. *AAAI Case-Based Reasoning Workshop*. Washington, D.C., July 1993, pp. 19-25.
- Kolodner, J. L. and Wills, L. M.: 1994. Case-Based Creative Mechanical Design: Selected Papers. GIT-COGSCI-94/14.
- Koton, P.: 1988. Reasoning about evidence in causal explanation. In *Proceedings of the 6th National Conference on Artificial Intelligence*. Cambridge, MA: AAAI Press/MIT Press.
- Maclean, A., Young, R. M., Bellotti, I. M. E., and Moran, J. P.: 1991. Questions, Options, Criteria: Elements of Design Space Analysis. *Human-Computer Interaction*, pp. 201-250.
- Maier, N. R. F.: 1931. Reasoning in humans II: The Solution of a problem and its appearance in consciousness. *Journal of Comparative Psychology*, 12, pp. 181-94.
- Maier, N. R. F.: 1970. *Problem Solving and Creativity: In Individuals and Groups*. Belmont, CA: Brooks/Cole Publishing Company, Study 14, pp. 162-175.
- Moorman, K. and Ram, A.: 1994a. A Model of Creative Understanding. In the *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 74-79. Seattle, Washington.
- Moorman, K. and Ram, A.: 1994b. Integrating Creativity with Reading: A Functional Approach. In the *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp. 646-651. Atlanta, GA.
- Navinchandra, D.: 1991. *Exploration and Innovation in Design: Towards a Computational Model*. New York: Springer-Verlag.
- Navinchandra, D.: 1992. Innovative Design Systems: Where are we, and where do we go from here?. Parts I and II. *The Knowledge Engineering Review* 7(3): 183-213 and 7(4): 345-362.
- Nersessian, N.: 1992. How Do Scientists Think? Capturing the Dynamics of Conceptual Change in Science. In *Volume XV: Cognitive Models of Science* of R. Giere, Ed., *Minnesota Studies in the Philosophy of Science* 15. Minneapolis: University of Minnesota Press.
- Nersessian, N.: 1993. In the Theoretician's Laboratory: Thought Experimenting as Mental Modeling. In *Proceedings of the 1992 Meeting of the Philosophy of Science Association*, Vol. 2, D. Hull, M. Forbes, and K. Okruhlik Eds. East Lansing Michigan: Philosophy of Science Association.
- Nersessian, N. & Greeno, J. (in process). Dynamic mental modeling in scientific reasoning, manuscript.
- Patalano, A., Seifert, C., Hammond, K.: 1993. Predictive encoding: Planning for opportunities. In *15th Cognitive Science Conference*, (pp. 800-805). Lawrence Erlbaum.
- Prabhakar, S. and Goel, A.: 1992. Performance-Driven Creativity in Design: Constraint Discovery,

- Model Revision, and Case Composition. In *Proceedings of the Second International Conference on Computational Models of Creative Design*. Heron Island, Australia.
- Ram, A.: 1994. AQUA: Questions that Drive the Explanation Process. In R.C. Schank, A. Ram, and C.K. Riesbeck (eds.), *Inside Case-Based Explanation*, Chapter 7. Lawrence-Erlbaum Associates.
- Ram, A., Wills, L., Domeshek, E., Nersessian, N., and Kolodner, J.: 1995. Creativity is in the Mind of the Creator. To appear in *Artificial Intelligence*. Also published as technical report GIT-CC-94/13.
- Schank, R.: 1982. *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press.
- Seifert, C., Meyer, D., Davidson, N., Patalano, A., and Yaniv, I.: 1994. Demystification of Cognitive Insight: Opportunistic Assimilation and the Prepared-Mind Perspective. In R.J. Sternberg and J.E. Davidson (eds.), *The Nature of Insight*. Cambridge, MA: MIT Press.
- Simina, M. and Kolodner, J. L.: 1995. Opportunity Recognition: A Design Perspective, submitted to *The Seventeenth Annual Conference of the Cognitive Science Society*.
- Sycara, K. and Navinchandra, D.: 1989. Representing and Indexing Design Cases. In *Proc. of Second Int. Conf. on Industrial and Engineering Applications of AI and Expert Systems*, Tullahoma, TN.
- Turner, S. R.: 1994. *MINSTREL*, Lawrence-Erlbaum Associates, Inc.
- Vosniadou, S., Tiberghien, A., and Mandl, H.: 1992. Modelling the learner: lessons from the study of knowledge reorganization in astronomy. In *Proceedings of the NATO Advanced Research Workshop on Knowledge Acquisition in the Domain of Physics and Intelligent Learning Environments*, pp. 101-110.
- Wallas, G.: 1926. *The Art of Thought*. New York: Harcourt Brace Jovanovich.
- Wills, L. M. and Kolodner, J. L.: 1994. Explaining Serendipitous Recognition in Design, *The Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA, pp. 940-945, 1994.
- Wills, L. M. and Kolodner, J. L.: 1994. Towards More Creative Case-Based Design Systems, in *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington, pp. 50-55.

The Role of Generic Modeling in Conceptual Change

Todd W. Griffith
Nancy J. Nersessian
Ashok Goel

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 853-9381
{griffith,nancyn,goel}@cc.gatech.edu

Abstract

The research presented here investigates what we hypothesize to be a central process in conceptual change in science: "generic modeling". We begin by discussing two notions of generic modeling. The first notion, generic mental modeling, is part of a cognitive-historical theory of reasoning productive in conceptual change, "constructive modeling" (Nersessian, 1995b, in press). This theory has its origins in a philosophical analysis of historical conceptual change in science. The second notion, generic mechanisms, are modeling strategies that form part of a computational theory, "adaptive modeling" (Bhatta & Goel 1993). This theory has its origins in a computational analysis of creative design. Using a constructive modeling interpretation of the reasoning exhibited in protocols collected by John Clement (1989) of a problem solving session involving conceptual change, we employ the resources of the theory of adaptive modeling to develop a computational model. Here we describe a piece of our analysis of the protocol to illustrate how our synthesis of the two theories is being used to develop a system, ToRQUE, for articulating and testing them both. The results of our research show how generic modeling plays a central role in conceptual change. They also demonstrate how such an interdisciplinary synthesis can provide significant insights into scientific reasoning.

Subject Area(s): Cognitive Science: Conceptual Change

Disciplinary areas: Computer Science: Mental Models, Model-Based Reasoning, Philosophy:
Philosophy of Science, Psychology: Analogical Reasoning, Mental Models.

Keywords: conceptual change, generic abstraction, model revision

Presentation Format: Talk

Note: First author is eligible for the Marr prize.

This research was funded in part by NSF Grant No. IRI-92-10925 and in part by ONR Grant No. N00014-92-J-1234. We thank John Clement for allowing us to use his protocol transcript and James Greeno for his contribution to developing our constructive modeling interpretation of it.

1. Conceptual Change in Science

In many instances, solving novel or difficult problems leads to conceptual change. Such conceptual change can range from minor changes in existing concepts to the radical kind of change one associates with "scientific revolutions". Some philosophers and psychologists have provided coarse-grained constraints for the processes of conceptual change (See Nersessian 1995a for an overview). To further our understanding, these modes of reasoning need elaboration and specification. One significant problem is how existing knowledge, often from remote domains, can be used in creating genuinely novel understandings. We contend that generic modeling of knowledge of properties, relations, principles, mechanisms, etc. play a key role in conceptual change in science. In the research discussed here we analyze the role of generic modeling in a problem solving protocol collected by John Clement (1989). Our analysis makes use of the cognitive-historical theory of constructive modeling (Section 3) to provide a conceptual analysis of the problem-solving session (Section 4). We then join this analysis with the computational theory of adaptive modeling (Section 5) that we believe provides the resources necessary to model the protocol as so analyzed. Together, the conceptual analysis and the AI theory provide sufficient constraints to implement an experimental system we call ToRQUE (Theory Revision through Questions, Understanding, and Evaluation)(Section 7).

2. The Clement Protocol

The problem posed in the Clement protocol is as follows:

"... a weight is hung from a spring. The original spring is replaced with a spring made of the same kind of wire; with the same number of coils; but with coils that are twice as wide in diameter. Will the spring stretch from its natural length more, less, or the same amount under the same weight? (Assume the mass of the spring is negligible compared to the mass of the weight.) Why do you think so?" (Figure 1 a & b)

In the study, subjects were asked to assess their confidence in their answer and in their understanding. We focus on one subject, S2, who changed his concept of a spring by incorporating the physical principle of torque into his understanding of how springs function.

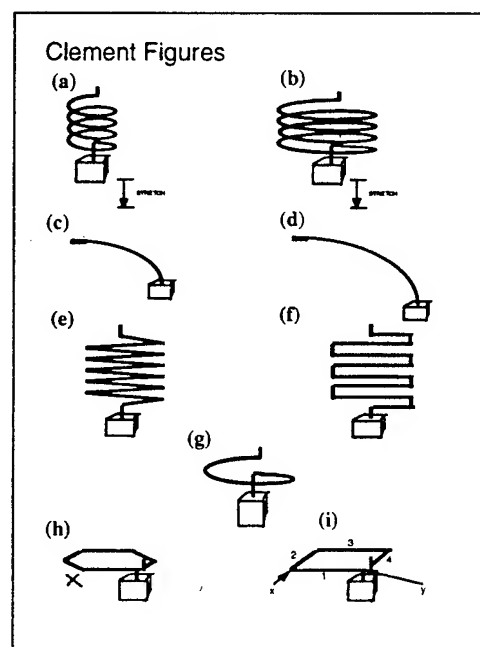


Figure 1

Unable to solve the problem directly, S2 began by reasoning that a spring when it is unwound is like a flexible rod (Figure 1c). He then reasoned that a spring of twice the diameter can be unwound into a longer rod, which will bend farther given equal force (Figure 1d). From this he concluded (correctly) that a spring of twice the diameter will stretch farther given equal force. S2, however, unlike most of the participants in the study, was not confident of this answer, and he was even less confident in his understanding of the problem. He noticed that a significant difference between the stretched spring and the bent rod is that the bent rod has a varying slope, while the spring has a constant slope, i.e., the space between the coils is uniform both before and after the spring is stretched. At this point S2 constructed the models that are the primary focus of our modeling effort (Figure 1e-i). These models were constructed based on salient differences between the spring and the flexible rod, and are designed to resolve what S2 regarded as an anomaly: the nonuniform slope of the bending rod (see Darden 1991). He eventually constructed a model of a hexagonal coil (Figure 1g) that led to the understanding that a spring maintains its constant slope through the twist of the coil wire during bending. The notion of torque was not present in S2's original model of spring, so we contend that S2's concept of a spring is changed in the problem solving process. Although we are modeling the whole protocol, given space limitations we will focus on just this final piece of reasoning and how we interpret it as exhibiting "generic modeling".

3. Constructive Modeling

Nersessian (1992, 1995a, in press) has argued that general modes of reasoning such as visual reasoning, thought experiment, analogy, and generic modeling play significant roles in scientific conceptual change. These various modes often are employed together in an iterative reasoning process we call "constructive modeling." Constructive modeling is a semantic process in which the models produced are proposed as interpretations of the target satisfying specific constraints. Figure 2 provides a schematic representation of such a process. Constructing

Constructive Modeling

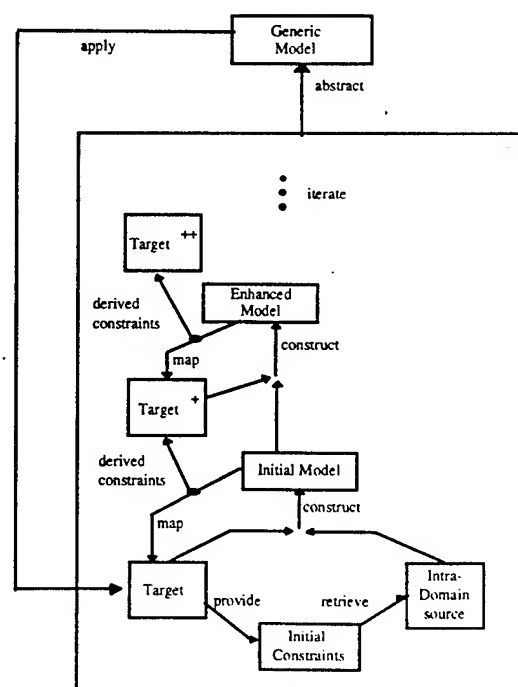


Figure 2

a model starts with properties and relations of a target system that serve as constraints to be satisfied by the initial model. A source domain satisfying some initial target constraints is retrieved. From this domain an initial analog model is retrieved or is constructed in the case where no direct analogy. This initial model - and each constructed model - serves as a source of additional constraints that interact with those provided by the target system to create an enhanced understanding of the target, in particular by making explicit further target constraints. The constraints can be supplied in different informational formats, including equations, texts, kinesthetic, diagrams, pictures, maps, and physical models. The model construction process involves different forms of abstraction (limiting case, idealization, generalization, generic modeling), constraint satisfaction, adaptation, simulation, and evaluation. Additional source domains may be called upon throughout the iterations. This cycle is repeated until a satisfactory representation of the target problem is achieved. This representation is a model of the same type as the target problem with respect to the salient target constraints. We interpret S2's reasoning to be a case of constructive modeling (Figure 2).

Clearly, to engage in constructive modeling the reasoner needs to know the generative principles and constraints for physical models in one or more domains. This is why analogy plays such a significant role in the constructive modeling process. On our account, the function of analogies is to provide constraints and generative principles for building models. This view is in contrast to the direct transfer view of most computational models (see for example Falkenhainer *et al.*, 1989; Holyoak & Thaggard 1989) Thus we view relations between domains in terms of the constraints they share. These constraints and principles may be represented in the different informational formats and knowledge structures that act as explicit or tacit assumptions employed in constructing and adapting models during problem solving. Since these constraints are domain-specific they need to be understood at a sufficient level of abstraction in order for retrieval, transfer and integration to be possible. We call this level of abstraction "generic".

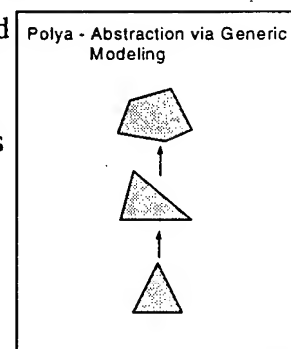


Figure 3

What we mean can easily be conveyed by looking at a simple example taken from Polya (1954). Polya considered two cases, abstracting from an equilateral triangle to a triangle-in-general and from it to a polygon-in-general (Figure 3). Loss of specificity is the central aspect of

this kind of abstraction process. Polya called the process "generalization" in mathematics, but we prefer to call it "generic abstraction" in order to distinguish it from the process of "generalization" in logic. The generic triangle is understood to represent those features that all kinds of triangles have in common. Although the figure entertained by the mind is specific, some of its salient features, the lengths of the sides and the degrees of the angles, must be taken by the reasoner to be unspecified. In contrast to this, a logical generalization from one equilateral triangle to all equilateral triangles maintains the specificity of these salient aspects of "equilateral". In abstracting from the generic triangle to the generic polygon, additional features are left unspecified, viz., the number of sides and the number of angles of the figure.

Generic modeling is a strategy that is commonly employed in solving physics problems. For example, in modeling a problem about a pendulum by means of a spring, the scientist understands the spring model as generic, that is, as representing the class of simple harmonic oscillators of which the pendulum is a member. We interpret much of the research in expert physics problem solving as demonstrating this (see for example Chi *et al.*, 1981). Generic modeling has also been shown to play an important role in design (see Stroulia & Goel 1992 and Bhatta & Goel 1993). Further, we believe it facilitates analogical retrieval, mapping, and adaptation in the model construction process. This is exemplified in the psychological literature by Holyoak and collaborators (see for example Gick & Holyoak 1983). Through the generic modeling process, knowledge from multiple domains is brought to bear on a problem and is capable of being transformed to such an extent that something truly novel is constructed, as is the case in conceptual change.

There are several ways in which we interpret generic modeling as playing a role in S2's constructive modeling process: generic abstraction is employed to create models that incorporate constraints from multiple domains; generic adaptation strategies are employed to make changes to models, such as topological transformations; and knowledge of generic mechanisms and principles is used in model construction and adaptation.

4. A Constructive Modeling Interpretation of S2's Reasoning

S2 spent considerable time considering his "physical imagistic intuition" (025)¹ about the slope of the bending rod. We begin at the point he claimed to have a visual experience that

¹ These numbers are line numbers from John Clement's original protocol. (see also Clement, 1989)

"expressed what [he was] thinking" (049) With the rod one "is always measuring in the vertical -- maybe somehow the way the -- the coiled spring unwinds, makes for a different frame of reference." (049) This insight would lead, though not immediately, to a model of the spring as an open horizontal (3-d) coil (Figure 1g). This part of the session generated a target constraint that was salient in this and the final two models (1e,i): coiling is in the horizontal plane.

At this point S2 was seeking to reconcile the rod (1c) and circular coil (1g) models. He achieved reconciliation by integrating the rod model with target constraints derived during the problem solving process: circularity, lying in the horizontal plane, and uniform distortion during stretching. S2 recognized that transmitting the force incrementally along the circle in the horizontal plane stretches it bit by bit, as though it had joints, but with even distribution. He now recalled an earlier idea of modeling a spring by means of wound square coils, i.e., that a "square is sort of like a circle". (117) We interpret him to mean that squares, considered generically are polygons and polygons approximate circles in the limit. He immediately considered bending up the rod into an approximation of the circle to create "a continuous bridge" between the two paradigmatic cases. We take this as his attempt to ascertain if a rod bent in a joint-like fashion in the horizontal plane and a circle bending under a force transmitted incrementally are of the same type with respect to the mechanism of bending. This interaction between the enhanced target (unfolding circle) and the initial source model (flexible rod) led to his constructing a series of generic polygonal models we have represented in Figure 4a.

S2 first drew a picture of a horizontal hexagon (Figure 1h) and saw immediately that the hexagonal model is a model of a different type from any considered before for how the constraints would interact in the dynamic case where the spring is stretched. S2's next statement described a simulation that provided a crucial insight: "Just looking at this [1h] it occurs to me that when force is applied here, you not only get a bend on this segment, but because there's a pivot here ['X' in 1h], you get a torsion effect -- around here." (121) He went on, "Aha! -- Maybe the behavior of the spring has something to do with the twist forces that might be the key difference between this [flexible rod], which involves no torsion, and this [hexagonal coil]." (122) Finally, S2 constructed the last

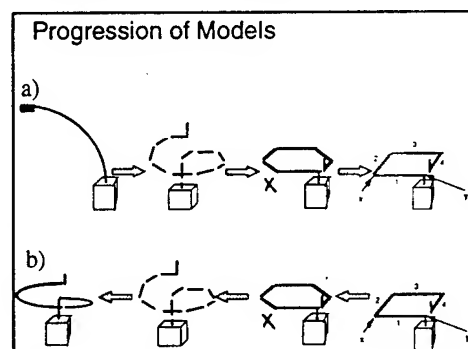


Figure 4

model, drawing a square coil (1i) in order to exaggerate the torsion effect and considered the possibility that torsion is what "stops the spring from -- from flopping." (126).

We interpret these steps in S2's problem solving as generic modeling of the relational structures and physical properties of the polygonal models. Both the hexagon and the square models incorporate features of the rod because the straight-line segments can bend. However, in this orientation any polygonal model will localize the torsion at the corners, so that the motion in stretching is that of twisting rather than bending at the joints. Thus there is torsion plus bending in this stretching process. The square coil model or the hexagonal coil model or any polygonal model will provide a generic model of the spring coil with respect to the mechanism of stretching. The key difference between the polygonal models (1g-i) and earlier models we have not discussed here (1e,f) is that in the former the bending segment does not have to change directions as it does in the latter models, where the bend cannot be spread out so as to occur continuously in the wire. When the wire is coiled in the horizontal plane, the bend is in the same relation to each piece and the springiness is distributed evenly, satisfying the target constraints. That the distribution of the twist would be even can be seen by extrapolating the polygon to the limit of a circle. Although these steps are not in the protocol, we interpret generic modeling to have enabled S2 to grasp immediately the move backwards from the square coil to the hexagon to intermediate extrapolations to the limit of the circular coil in which the torsion that is localized at the corners spreads itself out in such a way that it becomes a uniform property of the spring (Figure 4b).

5. Adaptive Modeling

Since we view constructing or designing models as central in conceptual change, we have chosen to start with AI theories of design for identifying language constructs and processing structures for building computational accounts of constructive modeling in science. In an independent line of research, Goel and collaborators have developed an AI theory of conceptual design of physical devices that views device design as model construction. This theory, called adaptive modeling, arose from work on the Kritik project (Goel 1991). A designer's comprehension of the functioning of a known device is represented in the form of a structure-behavior-function (SBF) model that provides a functional and causal explanation of how the structure of the device delivers its functions. New devices are designed by constructing SBF models for them, and new models are constructed by adapting the models of known devices. The

SBF models of the new device designs are verified through a form of qualitative simulation, and, if needed, revised.

Recent work along this line of research has led to a theory of creative conceptual design. This theory extends and expands Kritik's theory of adaptive modeling by incorporating analogical transfer as another family of adaptation strategies. It posits generic models for mediating the analogical transfer. In particular, it identifies two kinds of generic models: generic teleological mechanisms (GTMs) and general physical processes (GPPS) (Stroulia and Goel 1992; Bhatta and Goel 1993). A GTM specifies a pattern of functional and causal structure such as feedback while a GPP captures a pattern of behavioral and causal structure such as heat flow. The generic models are abstracted from the SBF device models of a known design situation, indexed by the functional/behavioral abstractions, and stored in memory. Given a new design situation, the stored generic models are accessed and instantiated to help create SBF models for the new situation. The IDEAL system (Bhatta & Goel 1993) instantiates the extended theory of adaptive modeling. Thus, depending on the design situation presented to it and its relation to the available knowledge, IDEAL can use different model adaptation strategies ranging from incremental revision of known SBF models within the problem domain to cross-domain analogical transfer of modeling knowledge in the form of generic models. The SBF theory of device comprehension and the adaptive modeling theory of solving design problems together provide us with the representation and processing structures for beginning to build a computational account of the constructive modeling reasoning process in science.

6. Synthesis of Theories

By itself "constructive modeling" provides an outline for a process of scientific reasoning that results in conceptual change. In order to acquire a more specific understanding we have been developing of a computer system based on the principles of adaptive modeling to model our interpretation of the Clement protocol. This collaborative effort engages a problem central to cognitive science as an interdisciplinary research field: How can theories from different disciplines be synthesized to provide a richer understanding of reasoning processes? And how might a synthesis be utilized to develop computational systems for experimentation? In this project we have a cognitive-historical theory of constructive modeling paired with the computational theory of adaptive modeling. The result of this pairing is that we are provided with 2 kinds of constraints

for the choices we make in modeling the system. The first are cognitive constraints drawn from a "cognitive- historical" synthesis of philosophical, historical, and psychological studies of human reasoning. These include both interpretive constraints for analyzing data and processing constraints in the form of coarse-grained commitments. The second are computational constraints drawn from computer science and theories of cognition which include tractability, inferencing capability, and representational adequacy. Thus the choices we have made in building the ToRQUE system garner support from both theories and the interaction between them. In the next section we justify some of the choices that we have made in the development of the ToRQUE system with respect to the computational and cognitive constraints of these theories. In so doing we pave the way for a new cognitive theory of scientific reasoning which draws upon aspects of both.

7. Computational Analysis

For S2's reasoning the choice of adaptive modeling is particularly apt computationally for two reasons: there is a good match between the SBF formalism and the physical systems in question (i.e. springs, flexible rods, etc.) and, more importantly, SBF representations provide significant benefits with respect to the kinds of inferences available, and the speed with which those inferences are carried out. The structure (S) of S2's initial model of a spring is clearly one of multiple coil components that interact with one another. This interpretation is supported by S2's simplifying the representation by reducing the spring to a single coil: "It occurs to me that a single coil of a spring wrapped once around is the same as a whole spring." (023) The inference is not that a coil is equivalent to a spring, but that it has the same basic function (F) as a spring, because in most respects a coil is not the same as a spring. (e.g. it does not look like a spring or have the same structure as a spring) This inference provides evidence that S2 used separate notions of function (F) and structure (S). A spring and a coil can be "the same" functionally while not being the same structurally or topologically. It also shows that S2 considered the spring as divided into multiple coil components.

The task that S2 completed involves assessing the behavior (B) of a particular physical system with regard to its structure (S). Given a particular property of the spring's structure, e.g. the diameter value, how will the behavior of the spring be affected? S2's attempt to solve this problem requires having a representation of the behavior in question or being able to generate one

quickly. One of the advantages of adaptive modeling is that the explicit storage of this behavior provides a significant computational advantage over the generation of the behavior. The kind of inferences that can be made given the stored behavior are also important. For example, when S2 noticed the difference (change in slope in the flexible rod vs. uniform slope in the spring), he did so because the behavior shows this difference to be salient. By separating structure, behavior, and function into separately analyzable units, the SBF formalism prunes away differences that are irrelevant to the task, and makes it easier to target areas of significant difference. Thus once the model is paired with a task it is possible to see the salient differences without being distracted by ontologically distinct kinds of differences.

Once S2 considered a single coil in place of an entire spring we see that he began to focus on the topological feature of circularity. At this point in the protocol he has already considered the behavioral and structural differences, and has made some adaptations with respect to these parts of the model. The difference between the behavior of the flexible rod and the spring provided the initial set of salient differences and the structural adaptation from many coils to one coil allowed S2 to focus his attention on what turned out to be the most important differences: circularity and orientation.

At this stage in the protocol ToRQUE's SBF model of a coil and the SBF model of a flexible rod each have a single component which has the function of providing a restoring force. Because "Structure" refers to components and the connections of components, the structures of two devices with a single similar component are necessarily the same. The topologies of these devices, however, may still be significantly different. That S2 addressed the differences in this order provides further support that SBF structures are a useful ontology for focusing inferences. Problems such as S2's that involve behavioral aspects of the physical system are handled best by focusing on behavioral differences first. Thus S2 is required to make use of the topological differences between the coil and the flexible rod, only after he has pruned away those differences which are presented by the behavior and structure.

Just as IdEAL uses generic teleological mechanisms (GTMS) for adapting models, ToRQUE uses generic topological transformation mechanisms (GTTMS) for adapting models. Here we describe the use of these mechanisms with respect to S2's reasoning in the final insight section interpreted in Section 3. In ToRQUE, the "Reduce-Repeating-Components mechanism" is used

to reduce the spring to a single coil (Figure 5c). The "Transform-Segment-to-Closed-Figure" and Transform-Planar-Orientation bend the rod into a coil (5d). We assume here with S2 that a coil "is a circle with a break in it". Figure 5e shows the progression of closed-figure transformations, which lead to the hexagonal coil, the discovery of torque, and the exaggeration of the effect by the square coil model. By adapting the coil from a circle to a polygon, S2 was able to introduce new components into the model structure. Each side of the square, e.g., could now be treated as a flexible rod component, but with the significant change in orientation that now makes for twisting rather than bending at the joints. Thus a small topological change can result in a fairly large behavioral change, making new knowledge available from which to make inferences.

The most important inference occurs in evaluating the square coil. S2 had recognized the generic physical principle (GPP) of torsion in the hexagonal coil and constructed the square coil to examine it. He was reminded of this principle because of the behavioral and structural similarities between the GPP and the polygonal models. In Section 3 we interpreted S2 as making a final series of inferences only implicit in the protocol that involve the generic abstraction of the square coil with respect to torsion. To be satisfied that he had solved the problem, he needed to hypothesize that if torsion is true of square coils, perhaps it is true of all coils and to make the appropriate extrapolation. ToRQUE incorporates the GPP into the circular coil model through the Transform-Discrete-to-Continuous GTTM, which depends upon a knowledge of limits which we know S2 possesses: A continuous shape such as a circle can be thought of as containing an infinite number of infinitesimally small segments. Figure 5(f) shows the transformations from the square coil back to

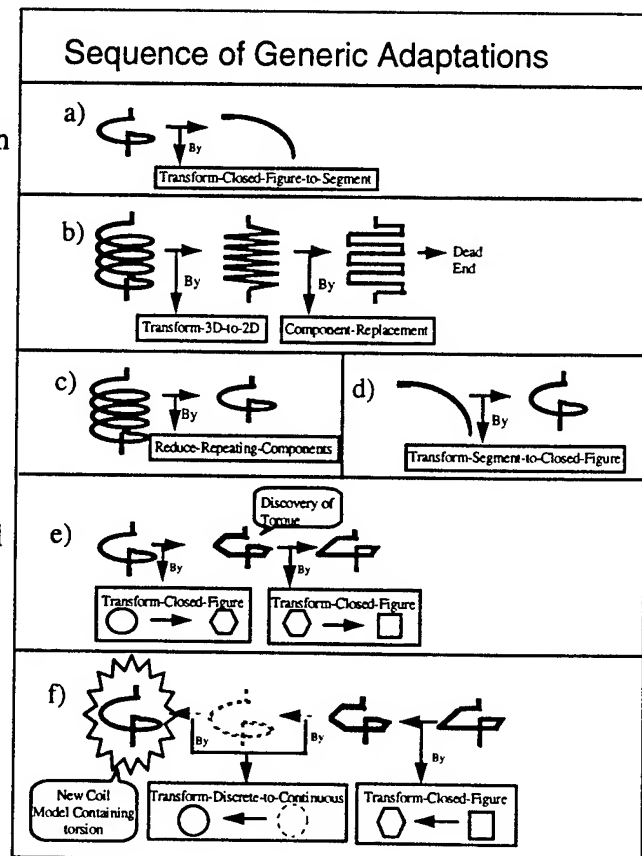


Figure 5

an adapted model of the circular coil that capture our interpretation.

8. Conclusion

Our conceptual analysis provides a plausible interpretation of S2's reasoning as relying significantly on generic modeling. Our computational analysis shows how generic modeling, through using models (e.g. generic coils), mechanisms (e.g., GTTM's), and principles (e.g., torque), can achieve conceptual change. Here we highlight two significant conclusions that show the synergy of our interdisciplinary collaboration:

- An important issue in generic modeling is how to make the right inferences at the right times. SBF models enable and constrain these inferences.
- In analyzing protocol and historical data there are places where the reasoning process is not explicit, as in the portion of S2's reasoning we examined here. Interpretations of these processes gain plausibility by their instantiation in computational systems such as ToRQUE that develop out of an interdisciplinary analysis of creative reasoning.

References

- Bhatta, Sambasiva R. & Goel, Ashok K. (1993) Learning Generic Mechanisms from Experiences for Analogical Reasoning. In *Proc. Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, Colorado, July 1993, pp. 237-242, Hillsdale, NJ: Lawrence Erlbaum.
- Chi, Michelene T. H., Glaser, P.J., & Glaser, R. (1981) Categorization and Representation of Physics Problems by Experts and Novices, *Cognitive Science*, 5, pp. 121-152.
- Clement, John (1989) Learning via Model Construction and Criticism: Protocol Evidence on Sources of Creativity in Science, In *Handbook of Creativity: Assessment, Theory and Research*, Glover, G., Ronning, R., & Reynolds, C. (Eds.), chapter 20, pp. 341-381. New York, NY: Plenum.
- Darden, Lindley (1991) Anomaly Driven Redesign of a Scientific Theory: The TRANSGENE.2 Experiments, Technical Report, Ohio State University.
- Falkenhainer, B., Forbus, K.D., and Gentner, D. (1989) The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41, 1-63. (University of Illinois Technical Report UIUCDCS-R-87-1361, July, 1987).
- Gick, M.L. & Holyoak, K.J. (1983) Schema induction and analogical transfer. *Cognitive Psychology*, 12(3), 306-355.
- Goel, Ashok, K. (1991) Model revision: A theory of incremental model learning. In *Proc. of the Eighth International Conference on Machine Learning*, pages 605-609, Chicago.
- Holyoak, K. & Thaggard, P. (1989) "Analogical Mapping by Constraint Satisfaction: A Computational Theory." *Cognitive Science* 13:295-356.
- Nersessian, Nancy J. (1992) How Do Scientists Think? Capturing the Dynamics of Conceptual Change in Science, In *Cognitive Models of Science*, ed. R.N. Giere. pp. 3-44. Minneapolis, MN: University of Minnesota Press.
- Nersessian, Nancy J. (1995a) Opening the Black Box: Cognitive Science and History of Science, *Osiris* 1995, 10:194-211.
- Nersessian, Nancy J. (1995b) Constructive Modeling in Creating Scientific Understanding, *Science & Education*, 4: 203-226.
- Nersessian, Nancy J. (in press) Abstraction via Generic Modeling in Concept Formation in Science. In *Idealization in Science*, M.R. Jones & N. Cartwright, eds. (Rodophi).
- Polya, G. (1954) *Induction and Analogy in Mathematics*, Vol. 1, Princeton University, Princeton.
- Stroulia, Eleni. & Goel, Ashok K. (1992) Generic Teleological Mechanisms and their Use in Case Adaptation, In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, 319-324, Lawrence Erlbaum, Hillsdale, N.J.

CONSTRUCTIVE MODELING IN SCIENTIFIC DISCOVERY

Nancy J. Nersessian
Todd W. Griffith
Ashok Goel

Georgia Institute of Technology
{nancyn,griffith,goel@cc.gatech.edu}

1. Background, Motivations, and Goals

Scientific progress involves a complex interplay between data about the phenomenon under study, languages for describing the phenomenon, and content theories of the phenomenon. In particular, when there is a good match between interpretations of the modeling constraints represented in the data and the available languages for modeling the constraints, then it becomes possible to develop more precise and detailed content theories. Our goal is to develop new content accounts of significant processes involved in scientific discovery. This paper represents one step in a research program that merges a cognitive-historical model of historical discovery processes, verbal protocols collected from scientists, and AI languages for describing, analyzing and modeling scientific discovery.

Bacon [Langley et al, 1987] is a good example of early AI systems that modeled scientific discoveries. Bacon uses the AI language of problem spaces, production rules and heuristic search for modeling the discovery of simple algebraic patterns in complex scientific data. Transgene [Darden, 1991a] is a more recent example of an AI system capable of scientific theory formation. It views a scientific theory as an abstract "device" and data-driven scientific theory formation as a kind of device "redesign." Transgene uses AI languages for device modeling for representing early theories of plant genetics in the form of a functional device model. Its redesign process is constrained by a prior analysis of the historical data on theory revision in plant genetics around 1900. As additional cases of scientific discovery from different domains are analyzed, and as new AI representation and processing languages are developed, it becomes possible to develop richer computational models of various aspects of scientific discovery.

We view scientific "discovery" as "design" or "construction" in the process of solving problems. This view links scientific understanding and explanation to problem solving and

representation through a reasoning process that we call "constructive modeling" (Nersessian 1992, 1995, in press). The quest for understanding, for possessing a satisfactory explanation of some phenomena are driving forces behind scientific discovery. We interpret Simon's claim that problem solving is representation as an expression of this aspect of scientific discovery. The problem solver represents and re-represents a problem until the answer is obvious, i.e. the solution is understood and a satisfactory explanation is in hand. Constructive modeling is an iterative reasoning process that employs analogical and visual reasoning and mental simulation to create and adapt representations of a target problem.

We first identified the reasoning process in a case of creative scientific problem solving that resulted in major conceptual innovation in physics: James Clerk Maxwell's construction of the electromagnetic field equations (Maxwell 1861-2; Nersessian 1984, 1989, 1992, 1993, in press). In the process of constructing the equations, Maxwell added a new explanatory device to the conceptual and analytical resources of the physics community, the field representation of forces. As we have discussed in great detail in earlier work, Maxwell derived the field equations by constructing a series of models embodying the pertinent physical and mathematical constraints. In the process he used multiple knowledge domains, including, electromagnetism, continuum mechanics, and machine mechanics and various informational formats, including equations, linguistic representations, diagrammatic representations. Once identified, similar modeling practices can be seen to be widespread in science and engineering. Newton's modeling of the motion of the moon by means of an object falling from a mountain and Rutherford and Bohr's modeling of the motion of an electron by means planetary motion provide comparable cases of historic discovery. Additionally, we have identified instances of constructive modeling in studies of technological invention (Gorman & Carlson 1990). Further, we believe much of the informal modeling exhibited by expert subjects in protocols of problem solving experiments conducted by cognitive scientists can be interpreted as instances of constructive modeling (See, e.g., Chi *et al.* 1981, Clement 1989, Larkin *et al.* 1980, Nersessian 1995).

The analysis of constructive modeling developed initially as part of a philosophical theory of conceptual change in science. Drawing from historical data we hypothesized that the kind of modeling practiced by Maxwell, Newton, and others is not simply an "aid" to scientific reasoning but that it is in fact a central mechanism of conceptual change. Since the modeling process

employs analogy, visual representation, and simulative reasoning we have investigated a cross section of cognitive science research in the areas of analogy, visual/diagrammatic reasoning, mental modeling, and expert/novice problem solving to assist in developing a cognitive-historical model of the process (Nersessian 1992, 1995). The theory views the constructive modeling process as highly productive of representational change because it requires synthesizing constraints from the problem domain with constraints drawn from one or more analogical sources into models that provide interpretations of the phenomena under investigation. Further, constructive modeling brings to bear on the problem knowledge that is represented in different informational formats: some knowledge of constraints is expressed in language and equations; some is implicit in visual representations; some is either explicit or implicit in the analogies employed. We hypothesize that one major advantage of reasoning via model construction is that the models enable scientists to perform mental simulations that embody and operate with explicit and implicit constraints. These simulations provide an interplay between formal and informal in construction and evaluation.

To explain how constraints from often disparate domains can be successfully integrated into a single model, the theory posits the central role of generic mental modeling in promoting analogical retrieval and transfer and in enabling model adaptation. The construction process continues until a model of the same type with respect to, e.g., the mechanism or process under investigation is constructed. This generic model can then be reapplied to the problem to provide a domain-specific solution. For example, Newton's generic model of projectile motion can be reapplied to the domain of planetary motion to determine that the planets move nearly in ellipses around the sun.

As part of a philosophical theory, constructive modeling is in the form of a cognitive-historical model which makes high level, coarse-grained representational and processing commitments that provide a framework for understanding representational change. We are developing a more precise understanding of this important reasoning process through interaction with the computational theory described in Section 4, beginning with a computational analysis of a problem solving protocol we interpret as exhibiting this process.

Since we view scientific "discovery" as "design," it follows that we should start with AI theories of design for identifying language constructs and processing structures for building

computational accounts of constructive modeling in science. In an independent line of research, we have developed an AI theory of conceptual design of physical devices that views device design as model construction. This theory, called adaptive modeling, arose from our work on the Kritik project [Goel 1989, 1991a, 1991b; Goel and Chandrasekaran 1989, 1992]. A designer's comprehension of the functioning of a known device is represented in the form of a structure-behavior-function (SBF) model that provides a functional and causal explanation of how the structure of the device delivers its functions. New devices are designed by constructing SBF models for them, and new models are constructed by adapting the models of known devices. The SBF models of the new device designs are verified through a form of qualitative simulation, and, if needed, revised. The Kritik project led to the identification of a set of general adaptation strategies for modifying the models of known devices and for revising the models of new designs.

Recent work along this line of research has naturally led us to a theory of creative conceptual design. This theory extends and expands Kritik's theory of adaptive modeling by incorporating analogical transfer as another family of adaptation strategies. It posits generic models for mediating the analogical transfer. In particular, it identifies two kinds of generic models: generic teleological mechanisms (GTMs) and general physical processes (GPPS) [Goel 1989; Stroulia and Goel 1992; Bhatta and Goel 1993, 1994]. A GTM specifies a pattern of functional and causal structure such as feedback while a GPP captures a pattern of behavioral and causal structure such as heat flow. The generic models are abstracted from the SBF device models of a known design situation, indexed by the functional/behavioral abstractions, and stored in memory. Given a new design situation, the stored generic models are accessed and instantiated to help create SBF models for the new situation. The IDEAL system [Bhatta, 1995] instantiates the extended theory of adaptive modeling. Thus, depending on the design situation presented to it and its relation to the available knowledge, IDEAL can use different model adaptation strategies ranging from incremental revision of known SBF models within the problem domain to cross-domain analogical transfer of modeling knowledge in the form of generic models. The SBF theory of device comprehension and the adaptive modeling theory of solving design problems together provide us with the representation and processing structures for beginning to build a computational account of the constructive modeling reasoning process in science.

As our first attempt at developing a computational model of constructive modeling in science we examine an expert problem-solving protocol obtained in a interview conducted by John Clement (1989). As we interpret this protocol the subject used constructive modeling to satisfy himself that his initial solution to a problem was the correct solution. This example is much more constrained than the historical cases of scientific discoveries, yet complex enough to require that we wrestle with many of the quite difficult modeling issues the historical discoveries present. We concur with Herbert Simon (Simon, 1981) in presupposing that the problem-solving strategies scientists have invented and the representational practices they have developed over the course of the history of science are sophisticated and refined outgrowths of ordinary reasoning and representational practices (Nersessian 1992). The solution to a specific problem may or may not be possible within the confines of existing conceptual and analytical resources in the problem domain. When the resources within the problem domain are inadequate, the scientist may have to bring additional resources from other domains or create new resources. "Scientific revolutions" are the most dramatic examples of this. In the protocol we analyze, the reasoning of the expert subject is within the domain of the given problem. But with this limitation, the reasoning manifests much of the complexity and power of constructive modeling. Modeling cross-domain reasoning naturally requires a much greater infusion of knowledge into our computational model. This is possible in principle but is feasible in practice only when the viability of the modeling framework has been demonstrated.

In Section 2 we present a theoretical analysis of the constructive modeling reasoning process; in Section 3, a conceptual analysis of the experimental protocol as a case of constructive modeling; in Section 4, a discussion of the AI theory of adaptive modeling that informs our computational model, and in Section 5, a computational analysis that we are developing of the experimental protocol.

2. Constructive Modeling

For AI to make substantial progress in modeling creative scientific reasoning processes it needs to start from a rich understanding of what these processes are in actual scientific practice. On our interpretation of the historical records, constructive modeling is a reasoning process that has produced significant representational change in science. Thus we begin by laying out the

framework of constructive modeling to provide at least a flavor of the nature of the representational and processing resources required of an AI system to perform this kind of scientific reasoning. This will assist in the argument we present later in the paper as to why we believe the choice of adaptive modeling theory appropriate for building a computational theory of the constructive modeling reasoning process.

Most computational approaches have been taking formal reasoning as the prototype for scientific thinking, as have traditional accounts of scientific reasoning developed in the philosophy of science and employed at least tacitly by historians of science. The Maxwell case provides a particularly salient example of the novelty of our interpretation. On the "standard" interpretation the models Maxwell constructed were at best "merely suggestive" (Heimann, 1970) at worst, an "unproductive digression" (Chalmers, 1986) or fraudulent representations cooked up after he had derived the field equations through formal reasoning (Duhem 1902, 1914). On the constructive modeling interpretation model construction constitutes the reasoning process through which he derived the equations. As will be discussed in Sections 4 and 5, it is the constructive modeling *interpretation* of such reasoning exhibited in the historical and protocol records that makes our choice of AI theory appropriate.

2.1. The Maxwell Case

Since we are beginning our computational analysis with the experimental protocol obtained by Clement, we will provide only a brief description of an historical case of constructive modeling. We outline the reasoning Maxwell used in constructing a mathematical representation of the field conception of electromagnetic forces so that we may refer to it in theoretical analysis in the following sections on constructive modeling.

Maxwell (1861-2) started his reasoning with the goal of constructing a unified mathematical representation of the production and transmission of electric and magnetic forces with a time delay; i.e., of constructing a field representation of electromagnetic phenomena. He began in a setting rich with conceptual and analytical resources, and significant experimental results. This environment provided, among other things, the initial constraints of the target problem: (1) electric and magnetic actions are at right angles to one another, (2) the plane of polarized light is rotated by magnetic action, (3) there is a tension along the lines of force, and (4)

there is a lateral repulsion between the lines of force. Using these constraints, Maxwell first retrieved an elastic fluid medium as an analogical source. Note that, although we today view electromagnetism and continuum mechanical phenomena as different domains, from Maxwell's perspective the initial model was constructed using an intra-domain source because he believed the electromagnetic medium ("aether") to be a continuum mechanical system. He then constructed an initial model satisfying constraints drawn from the domains of electricity, magnetism, and continuum mechanics. This model describes the electromagnetic field as a fluid medium composed of vortices and under stress (Figure 1 a). In the initial phase of the analysis Maxwell concentrated on a single vortex and how its stresses and strains under motion represents the effects of various magnetic phenomena (Figure 1 b).

In the next stage of the reasoning process Maxwell examined the dynamical interrelations between electricity and magnetism, i.e., electricity produces magnetism and vice versa. Thus, in the model he needed to examine the dynamical interactions among the vortices. But this presented him with a problem since the vortices are all spinning in the same direction and friction at the points of connection would make the system stop. Thus, at this point he sought a different mode of connection among them. He made a cross-domain analogy to machine mechanics and introduced "idle wheel particles" between adjacent vortices, creating a hybrid model (Figure 1 c). He derived the mathematical representations for the dynamical relations between current and magnetism by expressing these in terms of relations between the particles and the vortices. Finally, by endowing the vortices with elasticity and representing electrostatic polarization by elastic displacement of vortices, he was able to calculate the wave of distortion produced by the polarization. This completed the solution to the target problem solution in that he had a unified, quantitative representation of the continuous transmission of electromagnetic actions with a time delay.

Maxwell formulated the laws of the electromagnetic field by abstracting from the models what continuum-mechanical systems, certain machine mechanisms, and electromagnetic systems have in common. We call this process "generic modeling" in the analysis below. In their mathematical treatment, these common dynamical properties, relations, and functions are separated from the specific systems in terms of which they had been made concrete. Once he had abstracted these properties, relations, and functions he was in a position to reconstruct the

equations using generalized dynamics (Maxwell 1864). because he knew how to represent potential and kinetic energy in the medium. This analysis assumes only that the electromagnetic system is a generic "connected system" that possesses elasticity and thus energy.

Figure 6 provides a schematic representation of Maxwell's constructive modeling process. In the construction process the analogical source domains provided constraints that were integrated with electric and magnetic constraints to create and enhance imaginary mechanical models which then could be mapped to the electric and magnetic systems. From Maxwell's writing it seems clear that his representations of the vortex medium and the intermediate abstractions such as fly wheels are largely visual. Further, reasoning with the models appears to require that they provide simulations. In the paper itself, Maxwell provided an extensive set of instructions for how his readers should visualize and animate the models in their own reasoning.

2.2. Symbolic and Iconic Representations

To clarify our analysis we will follow C. S. Pierce (1931-1958) by distinguishing two general kinds of representation, "symbolic" and "iconic". Symbolic representations include linguistic expressions and equations composed of symbols. These represent a physical structure or process by making propositional claims about it. Symbolic representations are interpreted as referring to objects, properties, and relations descriptively and can be evaluated as being true or false. Iconic representations include such things as diagrams, models, and gestures. Iconic representations are interpreted as representing objects, properties, and relations demonstratively and can be evaluated as being accurate or inaccurate. A model, thus, represents a physical structure or process by having surrogate objects with properties, relations, behaviors, or functions that are in correspondence to it.

Symbolic and iconic representations support reasoning in different ways. They enable different kinds of operations that can be applied to the representations. Operations on symbolic expressions include the familiar operations of mathematics and logic. These are truth preserving if the symbols are interpreted in a consistent way and the properties they refer to are stable in the environment. Additional operations can be defined that support reasoning in limited domains provided the operations are consistent with constraints that hold in the domain. Operations on iconic representations involve transformations of the representations that change their properties

and relations in ways consistent with the constraints of the domain (See, e.g., Hegerty 1992 on pulley systems). Significantly, transformational constraints represented in iconic representations may be implicit, e.g., a person can simulate what happens when a rod is bent without having an explicit rule that says what happens, such as "given the same force a longer rod will bend farther."

A simulation with an iconic representation involves constructing a model and running it, that is, produce new states. Constructing a model requires understanding the constraints governing the kinds of entities in the model and the possible structural, causal, and functional relations among them. Running a simulation requires understanding the constraints governing how those kinds of things behave and interact and how the relations can change. In a human reasoner the mental apparatus puts together constraints to achieve a causally coherent unfolding. Performing a simulation with a model supports inferences because the simulation complies with the same constraints as the system it represents. Further, changing the conditions of a model supports inferences about differences in the ways that a system behaves. Knowing a model means already having some facility with the constraints that are needed to construct and run it. A simulation creates new states of the system being modeled, which in turn create or make evident new constraints. A significant part of the creative process in scientific discovery is determining appropriate constraints for models in a domain.

2.3. Reasoning via Constructive Modeling

"Constructive modeling" is an iterative reasoning process of constructing and running iconic or mixed representations of a target problem. It is a semantic process in which the models produced are proposed as interpretations of the target satisfying specific constraints. Figure 4 provides a schematic representation of the process. Constructing a model starts with properties and relations of a target system that serve as constraints to be satisfied by the initial model (e.g., Maxwell's initial electromagnetic constraints). A source domain satisfying some initial target constraints is retrieved (e.g., the continuum mechanics of elastic fluids). From this domain an initial analog model is retrieved or is constructed in the case where no direct analogy exists (e.g., initial vortex model). This initial model - and each constructed model - serves as a source of additional constraints that interact with those provided by the target system to create an enhanced understanding of the target, in particular by making explicit further target constraints (e.g.,

mathematical constraints governing the interaction of electricity and magnetism). The constraints can be supplied in different informational formats, including equations, texts, kinesthetic, diagrams, pictures, maps, and physical models. The model construction process involves different forms of abstraction (limiting case, idealization, generalization, generic modeling), constraint satisfaction, adaptation, simulation, and evaluation. Additional source domains may be called upon throughout the iterations (e.g., machine mechanics). This cycle is repeated until a satisfactory representation of the target problem is achieved. This representation is a model of the same type as the target problem with respect to the salient target constraints (e.g., the vortex - idle wheel system and the magnetism and electricity systems are the same type with respect to dynamical structure).

Constructive modeling draws on the conceptual, material and analytical resources of the community in which the problem solver is situated. Concepts supply families of constraints that provide resources for constructing models. The resources of concepts include both their constituent constraints and the conventions of representation of those constituents developed by communities of practice for referring to properties, relations, and regularities of systems discussed in terms of those concepts. The material environment provides such resources as iconic representations that are drawn, gestured, or otherwise constructed, symbolic representations that are written or spoken, and experimental equipment. The analytical resources of a community include various modeling tools that include, among others, mathematics, diagrammatic forms of representation,, and computer simulations.

Clearly, to engage in constructive modeling the reasoner needs to know the generative principles and constraints for physical models in one or more domains. This is why analogy plays such a significant role in the constructive modeling process. On our account, the function of analogies is to provide constraints and generative principles for building models. Thus we view relations between domains in terms of the constraints they share. These constraints and principles may be represented in the different informational formats and knowledge structures that act as explicit or tacit assumptions employed in constructing and adapting models during problem solving. Since these constraints are domain-specific they need to be understood at a sufficient level of abstraction in order for retrieval, transfer and integration to be possible. We call this level

of abstraction "generic". "Generic modeling" plays a central role in facilitating analogical retrieval, mapping, and adaptation in the model construction process.

2.4. Generic Modeling

There are several ways in which generic modeling plays a role in the constructive modeling process: generic abstraction is employed to create representations that incorporate constraints from multiple domains; generic adaptation strategies are employed to make changes to models, such as topological transformations; and knowledge of generic mechanisms and principles is used in model construction and adaptation.

Consider the generic abstraction process. To integrate knowledge from a wide range of sources into a single a model requires a kind of abstraction in which the various representations function with some of their features unspecified. In reasoning, e.g., about a triangle, one often draws or imagines a concrete representation. Indeed as Kant has pointed out, one cannot imagine a "triangle in general" but only some specific instance of a triangle. However, in considering what it has in common with all triangles, humans have the ability to view the specific triangle as lacking specificity in its angles and the sides. That is, when the reasoning context demands it, the interpretation of the specific iconic representation can be as generic. In viewing a specific representation generically, one takes it as representing features common to a class of phenomena. Generality in representation is achieved by interpreting the components of the representation as referring to object, property, and relation *types* rather than *tokens* of these.

What we mean can be more easily conveyed by looking at a simple example taken from Polya (1954). Polya considered two cases, abstracting from an equilateral triangle to a triangle-in-general and from it to a polygon-in-general (Figure 3). Loss of specificity is the central aspect of this kind of abstraction process. Polya called this process "generalization" in mathematics, but we prefer to call it "generic modeling" in order to distinguish it from the process of "generalization" in logic. The figures presented are concrete, but the abstracted geometrical models are "generic". The generic triangle represents those features that all kinds of triangles have in common. The generic triangle has some of its salient features unspecified: the lengths of the sides and the degrees of the angles. The generic triangle is neither isosceles, scalene, nor equilateral. In contrast, a logical generalization from one equilateral triangle to all equilateral

triangles maintains the specificity of these salient aspects of "equilateral". In abstracting from the generic triangle to the generic polygon, additional features are left unspecified, viz., the number of sides and the number of angles of the figure.

Maxwell's introduction of idle wheels particles into the vortex-fluid medium provides an especially good instance of generic abstraction on the interpretation provided by constructive modeling (Figure 2). First, Maxwell abstracted the generic model of spinning wheels from his initial model of the electromagnetic medium described as vortices (A). The generic model of spinning wheels reminded him of specific mechanical systems containing machine gears (B). He noticed the analogy between the vortices and the gears (C) but how this analogy would provide a new mode of connection for the vortices was not yet clear to him. Next, from the model of the machine gears, he abstracted the generic model of fly wheels (D), and then further abstracted the fly-wheel model into a model of dynamically smooth connectors (E). Finally, he instantiated the generic model of smooth connectors in his model of the electromagnetic field in the form of idle wheel particles (F), where the instantiation is guided by both the analogous case of fly wheels (G) and constraints of the continuum mechanical system. A detailed analysis of generic modeling in Maxwell's analysis can be found in Nersessian (in press).

Generic modeling is a strategy that is commonly employed in solving physics problems. In modeling a problem about a pendulum by means of a spring, the scientist understands the spring model as generic, that is, as representing the class of simple harmonic oscillators of which the pendulum is a member. We interpret studies of expert physicists' problem-solving practices carried out by cognitive psychologists as exhibiting generic modeling as a means of subsuming a problem under a class of problems (See, e.g., Chi *et al.* 1981; Larkin *et al.* 1980, Clement 1989). Generic modeling could account for the expert's ease of transfer among problems, which is lacking in novice problem solving..

As we have said, we interpret the constructive modeling process as requiring the ability to represent, view, and utilize information generically. It is an open question in need of empirical investigation as to whether generic representations are stored in the human memory or whether specific representations are called upon and viewed as generic when the reasoning process demands. In either case, expertise seems to play a significant role in having facility with generic modeling. This is supported by the research of Mary Gick and Keith Holyoak (1980) which

provides evidence that in the process of analogical problem solving - especially with repetition - people create an abstract representations, or "schema", that facilitates transfer to a new problem.

Once identified as such, we can see how the practice of generic modeling has played a significant role in the development of modern science. It was only by generic modeling, e.g., that Newton could see the commonalities among the motions of planets and of projectiles which enabled his formulating a unified mathematical representation of motion. The generic model represents what is common among the members of classes of physical systems. Newton's inverse-square law of gravitation abstracts what the motion of a projectile and a planet have in common. The inverse-square-law model continued to served as a generic model of action-at-a-distance forces through the 19th century for those who tried to bring all forces into the scope of Newtonian mechanics.

In our philosophical theory of conceptual change, generic modeling provides an explanation of many instances of creativity. One puzzle about radical conceptual change has always been that given the implausibility of *ex nihilo* creation, how can fundamentally new conceptual structures emerge since we must always draw from existing representations? Generic abstraction makes it possible to bring knowledge from multiple domains to bear on a problem and have it transformed to such an extent that something truly novel is constructed. Our earlier work on Maxwell shows this in some detail (See Nersessian 1992, in press). Through using Newtonian mechanical systems as the sources of analogies, Maxwell abstracted generic representations of objects, processes, and mechanisms from these analogies and constructed a series of models satisfying salient electromagnetic constraints. The generic representations, when applied to the class of electromagnetic systems yield the laws of a non-Newtonian dynamical system.

3. The S2 Protocol

Our case study derives from problem solving protocols taken from an expert subject during a laboratory experiment designed by John Clement (1989). Clement's own analysis of S2's reasoning focuses on a process he calls modeling via "bridging analogies". He characterizes this process as one in which the subject "produces models via a successive refinement process of hypothesis generation, evaluation, and modification or rejection" (p.358). It is the specific nature of the construction and "successive refinement" process that leads to our interpretation of S2's

reasoning as a form of constructive modeling. The modeling process consisted of integrating constraints drawn from multiple sources, i.e., springs, bending rods, and geometry, into a series of models. During the process he spoke of visualizations, drew diagrammatic representations, and used gestures to simulate dynamical processes to be performed with the models and to provide a three dimensional perspective. Throughout the process, as Clement has argued, S2 critically evaluated the plausibility of the models and the inferences he drew from them. Figure 5 provides a schematic representation of S2's reasoning as constructive modeling.

According to Clement, S2 was a computer scientist who had some training in physics. He had also passed comprehensive examinations in mathematics in the area of topology, which is highly significant on our interpretation of the protocol session. S2's problem solving is embedded in a community of background knowledge, including analytical practices and conceptual resources. Some of that knowledge will be made explicit in our analysis of the constraints figuring in the modeling process.

On our interpretation, S2 began the problem-solving session with the understanding that the stretch of a spring is due to its flexibility. Through the constructive modeling process he derived a new understanding that a spring maintains constant slope when stretched through both twisting and bending. So, although this is a more modest outcome of constructive modeling that evidenced in historical cases of scientific discovery, for S2 it was an instance of highly creative problem solving. For S2 to find a satisfactory explanatory model for the problem solution, he had to construct a novel representation for himself of how a spring works.

In the following conceptual analysis we interpret S2's reasoning, focusing on the constraints and their origin at each stage, productive instances of generic modeling, and the geometrical insights that underlie the model construction and adaptation process. It is these features of the process that have guided our selection of the AI theory of adaptive modeling as the vehicle for exploring constructive modeling computationally.

3.1. Initial Problem Formulation

The problem to be solved is "a weight is hung from a spring. The original spring (Figure 7 a) is replaced with a spring made of the same kind of wire; with the same number of coils; but with coils that are twice as wide in diameter (Figure 7 b). Will the spring stretch from its natural

length more, less, or the same amount under the same weight? (Assume the mass of the spring is negligible compared to the mass of the weight.) Why do you think so?"

We assume the initial constraints provided by the target problem are:

- springiness
- constant number of coils
- variable coil width
- negligible mass
- constant added weight

3.2. Model 1: Flexible Rod

S2, along with a number of other subjects, quickly retrieved an initial analogy with a flexible rod (Figure 7c, d). S2 stated that stretching the spring coil to the limit would produce a flexible straight rod, which he later refers to as a "straightened spring". (005) We take this as a proposal that for the purposes of the problem, S2 assumed the rod and the spring are of the same type; i.e., they can be subsumed under a generic model of flexible wires. As a model, in the static case, the rod satisfies the constraints:

- variable coil width -> variable rod length
- negligible mass = negligible mass
- constant number of coils can be subsumed under variable length

In the dynamic case the mapping of the constraints would be:

- springiness -> flexibility
- constant weight = constant weight

For many subjects, including S2, this led directly to the correct conclusion that the wide spring would stretch further, i.e., under the dynamic condition of bending the rod under a constant weight, increasing the length of the rod would increase the amount it would bend. S2, however, expressed immediate dissatisfaction with this model because he saw that the rod "would -- droop like that [we assume a gesture] and its slope would steadily increase as you -- went away from the point of attachment whereas in a spring the slope of the spiral is constant." (005) That is, S2 recognized a new constraint that slope of the stretched spring is constant. We consider it plausible that S2's physics knowledge enabled a mental simulation of a bending rod that supported the inference that its slope is greater at positions closer to the weight. He later drew pictures to

explore this phenomenon. Additionally, the slope of a rod becomes greater as the rod increases in length. So, in the dynamic case:

- for the spring, slope = constant
- for the rod, slope = increasing with distance from attachment point

3.2.1. Enhanced Target Understanding (Target*)

S2 began to question his hypothesis that the rod provided model of the same type as the spring. He first expressed uncertainty that his intuition about the spring was correct, and spent several minutes trying to convince himself of it. This led him to the correct conclusion that the wider spring would stretch further, but not to an explanation for why this would be so. He stated that the conclusion came from imagining what would happen to the wider spring as it stretched, based in the "kinesthetic sense that somehow a bigger spring is looser." (005)

During the time it took to convince himself of the intuition about the constant slope of the spring, he drew pictures of rods and springs "to help fix [them] visually".(007) His drawing of the rod has it rotated at approximately forty-five (later ninety) degrees in the vertical plane. From this perspective, a rod fixed at the top would clearly bend with increasing slope ("flop") under a constant weight. The spring, on the other hand, would be free to "pivot" and "something about that sustains the angle...of the coil." (021)

S2 now had the understanding that springiness is different from flexibility ("bendiness"). Thus, for the purposes of this problem, the rod cannot provide a generic model of the spring because it violates the derived constraint for the dynamic case. That is, although S2 did not state this explicitly, we assume that he understood that if two objects are of the same type regarding a mechanism of distortion, they should be alike in whether the distortion is or is not uniform throughout the extent of the object. At this point, then, he added the derived constraint:

- spring slope remains constant during stretching under weight

His enhanced understanding of the target problem led him to express the need for an explanatory model of how a spring works before he could be satisfied with his answer.

3.3. Geometrical Insight

Before constructing Model 2 he generated two intuitions that would later be used in finding the satisfactory model. First, he noted that "a single coil wrapped around is the same as a whole spring" (023). We interpret this as a proposal that the single coil provides a model for the spring. In this context he rejected the proposal because it led him back directly to the rod model (a "straightened spring" coil). Second, he speculated that coiling is an accidental feature which led to the insight "surely you could coil a spring in squares, lets say, and it wouldn't -- it would behave more or less the same." (023) Later and in a different context these intuitions would lead to the critical insight that torsion is what keeps the spring slope constant.

In the present context, the second insight led to two unsatisfactory models that are hybrid modifications to the flexible rod model. These models derive from an attempt to integrate the constant slope constraint with the constraint that a spring is coiled.

3.4. Model 2: Zigzag Wire

In Model 1 S2 had assumed that coiling was "an accidental feature of the situation" (023) in the target case, but thinking about the spring as coiled in squares led to considering it as a possibly salient constraint:

- coiling (i.e., repeating coiled segment)

S2 stated "Ah! From squares, visually I suddenly get a kind of zigzag spring rather than a coiled spring" (023) and drew a picture of the zigzag model (Figure 7e). He then stated that the drawing was to be viewed as a "2-dimensional spring", not as a "profile of a 3-dimensional spring." We interpret this and his drawing to indicate that he was making an adaptation to the spring that gave it the form of a series of repeating rod segments. He endowed this model with the constraints:

- rigid bars
- bending at joints

The model is an attempt to satisfy the possible constraint, in that

- coiling -> zigzagging (i.e., repeating rod segment)

But he quickly saw that in the dynamic case it fails to satisfy his enhanced understanding of the target constraint of constant slope.

3.4.1. Enhanced Target Understanding (Target ⁺⁺)

In making the inference that the zigzag model does not satisfy the slope constraint, S2 appears to be drawing on physics knowledge:

- for the zigzag, the springiness is at the joints
- for the spring, the springiness is distributed along it

S2 understands bending at the joints is incompatible with uniform distribution of the stretch. This violation of type is a version of the one used to reject the rod model, i.e., the distortion of stretching is uniform throughout the helical spring, but is not in the zigzag model. Thus, the model and the spring cannot be alike with respect to the mechanism of stretching.

For S2 the target problem now contains the additional constraint:

- in a spring there is a uniform distribution of the stretch

3.5. Model 3: Rigid Connector Wire

Model 3 is constructed by attempting to integrate the target constraint of distributed springiness with Model 2. S2 claimed to visualize a potentially satisfactory mechanism by making the rods flexible and connecting them via small rigid rods (Figure 7f). He expressed imagining "kinesthetically" (025) pulling it, seeing that it stretches and bounces, as does a spring. However, if they are to be of the same type, it should be possible to consider just one segment, just as one can consider a single segment of the spring. But one segment is similar to the single rod and that led him back to thinking about why Model 1 will not work.

3.6. Geometrical Insight

S2 next spent considerable time considering his "physical imagistic intuition" (025) about the slope of the bending rod. He first called on mathematical knowledge of limits to consider what would happen by varying placement of weight along the rod and (after taking a phone call) "imagined" moving the weight along both the rod and the spring and drew pictures of what he

was imagining . This was followed by a lengthy period of considering whether a spring with wider coils is the same as a longer spring, i.e., one with more coils. At the end of this reasoning segment he expressed more confidence in his answer, but he was still bothered by why the constant slope constraint of the spring is violated by straightening it into a rod.

At that point he claimed to have a visual experience that "expressed what [he was] thinking" (049) With the rod one "is always measuring in the vertical -- maybe somehow the way the -- the coiled spring unwinds, makes for a different frame of reference." (049) This insight would lead, though not immediately, to Model 4, an open horizontal (3-d) coil (Figure 7g). First he again reassured himself that the spring has constant slope, calling this time on mathematical knowledge of differentials. He then noted that straightening the spiral of the spring vertically, as in the rod model, removes the curvature and observed that "all its curvature is sort of horizontal -- or near horizontal as the coils curl around" (055). He then spent some time thinking about the geometry of unwinding the coil and the differences between spirals and helices, drawing a picture of a spiral and comparing it with his earlier drawing of unwinding springs. Clement records that during this process S2 traced a circle about 7 inches in the air in front of himself and claimed to "imagine a coil -- a circle with a break in it. He drew (Figure 7g) an open single coil.

3.6.1 Enhanced target Understanding (Target ⁺⁺⁺)

This geometrical insight part of the session generated a target constraint that was salient in constructing Model 4:

- coiling is in the horizontal plane

We hypothesize that the open horizontal coil enabled a different kind of mental simulation, in which simulating a downward pull produced a simulated vertical gap between corresponding points on a coiled wire. S2 then fixed his attention on that drawing and drew a straight line, commenting that the circle and the line are "paradigmatic" (117). We interpret him to understand that they are generic abstractions that capture what he now takes to be salient about the cases of the wound-up spring and the unwound spring, respectively. However, drawing the line as bending again revealed the increasing slope in that case. This reinforced his understanding that the rod and the spring coil are not of the same type with respect to the mechanism of bending.

At this point in the session S2 expressed the need to think about the problem "radically different way" (117) and decided to focus on the fact that the coils of a spring are circular. In the process he entertained as possibly salient the target constraint:

- the coils are circular in form

3.7. Model 4: Polygonal Coil

This was not such a "radically different" way of thinking about the problem since at this point S2 was seeking to reconcile the rod and circle models he had considered earlier. He achieved reconciliation by integrating the derived target constraints of circularity, of lying in the horizontal plane, and of uniform distortion with the rod model. This attempt at integration led to the key insight that there is twisting in addition to bending and this is what accounts for the constant slope of the spring.

S2 recognized that transmitting the force incrementally along the circle in the horizontal plane stretches it bit by bit, as though it had joints, but with even distribution. At this point he recalled his earlier idea of modeling a spring by means of wound square coils, i.e., that a "square is sort of like a circle". (117) We infer him to mean that they are of the same type: squares, considered generically are polygons and polygons approximate circles in the limit.

S2 immediately considered bending up the rod into an approximation of the horizontal circle to create "a continuous bridge" between the two paradigmatic cases. We take this as his attempt to ascertain if a rod bent in a joint-like fashion in the horizontal plane and a circle bending under a force transmitted incrementally are of the same type. This interaction between the enhanced target (unfolding circle) and the initial source model (flexible rod) led to his constructing a series of polygonal models (Figure 16 a).

S2 stated that for the purposes of the spring problem, circles and polygons (he noted triangles, squares and hexagons) cannot be that much different, i.e., are of the same type. He first drew a picture of a horizontal hexagon (Figure 7h) and saw immediately that the hexagonal model is a model of a different type from any considered before for how the constraints would interact in the dynamic case. S2's next statement described a simulation that provided a crucial insight: "Just looking at this (Figure 7h) it occurs to me that when force is applied here, you not only get a bend on this segment, but because there's a pivot here (pointing to 'X' in Figure 7h), you get a

torsion effect -- around here." (121) That is, in the horizontal plane, the force from the weight would create twisting at the joints - torsion - as the hexagon unfolds. He went on, "Aha! -- Maybe the behavior of the spring has something to do with the twist forces [Clement recorded S2 moved his hands as if twisting an object]. That's a real interesting idea -- That might be the key insight -- that might be the key difference between this [flexible rod], which involves no torsion, and this [hexagonal coil]." (122) Finally, S2 constructed the last model, drawing a square coil (Figure 7i) in order to exaggerate the torsion effect and considered the possibility that torsion is what "stops the spring from -- from flopping." (126).

We hypothesize that by employing generic modeling of the properties of the polygonal models S2 became convinced that unlike the rod model, these satisfy the constant slope constraint. Our interpretation of his reasoning process is as follows. Both the hexagon and the square models incorporate features of the rod because the straight-line segments can bend. But because of their orientation, any polygonal models will localize the torsion at the corners, so the motion in stretching is that of twisting rather than bending at the joints. So there is torsion plus bending in the stretching process. Thus, the square coil model or the hexagonal coil model or any polygonal model provides a generic model of the spring coil with respect to the mechanism of stretching. The key difference between the polygonal models and the zigzag and rigid connector models is that in the former the bending segment does not have to change directions as it does in the latter models, where the bend cannot be spread out so as to occur continuously in the wire. When the wire is coiled in the horizontal plane, the bend is in the same relation to each piece and the springiness is distributed evenly, satisfying the target constraints. The distribution of the twist would be even as seen by extrapolating the polygon to the limit of a circle. Moving backwards from the square coil to the hexagon to intermediate extrapolations to the limit of the circular coil, the torsion that is localized at the corners spreads itself out in such a way that it becomes a uniform property of the spring (Figure 16 b).

S2 concluded the session by expressing that "I feel I have a good model of sp -- of a spring -- Now I realize the reason a spring doesn't flop is because a lot of the springiness comes from torsion effects rather than from bendy effects." (132) He went on to answer the problem. If the width of a coil is doubled, the increase in bending would also increase the torsion. Although the answer remains unchanged, i.e., the spring with wider coils will stretch further, the

understanding S2 has of a spring is considerably altered. With the spring-as-rod model, "springiness" is equated with bending. With the spring-as-horizontal-square model (or polygon), "springiness" is bending plus torsion.

Clearly, S2's representation of a spring has changed and his constructive modeling process is generative of that change. The non-constant slope of the bending rod model cued S2 about the additional target constraint that the stretched spring has a constant slope. The zigzag and rigid connector models directed him to focus on the circular nature of the spring's coils and that they lie in the horizontal plane. And, most critically, simulating bending in the horizontally rotated segmented hexagonal coil model led him to recognize that there is an invisible twist distributed along the coils of a spring, keeping the slope constant when it is stretched.

4. Adaptive Modeling

So far in this paper we have developed the hypothesis of constructive modeling as a novel interpretation of a significant reasoning process found in many historical cases of scientific discovery. In addition, we have analyzed S2's reasoning in Clement's protocol as one example of constructive modeling in creative scientific problem solving. We now seek to understand this case in terms of a computational theory of conceptual design of physical devices. We believe this computational theory, called adaptive modeling, offers an AI language for developing describing, analyzing and modeling scientific discovery processes of the kind manifested in S2's reasoning under the constructive modeling interpretation of the data.

The computational theory of "adaptive modeling" takes its name from the perspective it adopts on conceptual device design. Conceptual design generally refers to the preliminary phase of the design process. The problem-solving task in this phase takes a specification of the functions of the desired device as input. It has the goal of giving a high-level specification of a structure for the device as output, where the structure can deliver the desired functions. The perspective Adaptive Modeling adopts on device design is characterized by two views. First, it views device design as model construction. Thus the output of the problem-solving task is not just the device structure but a device model that specifies how the designed device is intended to work, how its structure delivers the functions desired of it. Second, it views device design as an evolutionary process in which new device models are constructed by adapting the known models of existing

devices. Depending on the differences between the specifications of the desired functions and the functions delivered by the known devices, adaptation strategies can range from revising the model of a known device, to generic modeling and use of generic models for transferring design patterns from one design problem to another.

Kritik and IDEAL are operational knowledge systems that instantiate the theory of adaptive modeling, enable experiments with it, and provide well-defined AI languages. Built in the late eighties, Kritik integrated case-based and model-based reasoning for modeling evolutionary design of simple physical devices [Goel 1991a; Goel 1992; Goel and Chandrasekaran 1989, 1992]. The key idea in the Kritik project was that evolutionary design involves both past design experiences (i.e., cases) and comprehension of how devices work (i.e., models) in creating new designs, that while the high-level process of design is largely case-based, device models give rise to both the vocabulary and the strategies for addressing the different tasks in the case-based process. The specific hypothesis in the Kritik experiments was that since the design task is a function \rightarrow structure mapping, the inverse structure \rightarrow function map of old designs may guide the adaptation of an old design to achieve a new functional specification. The structure \rightarrow function map of a device design in Kritik is specified as a Structure \rightarrow Behavior \rightarrow Function model. The SBF model of a device explicitly specifies the structure and the functions of the device as well as its internal behaviors that explain how the device functions are composed from the functions of its structural components. The behavior thus mediates between function and structure: it captures teleological and compositional knowledge of a device, and provides a functional and causal explanation of the how the structure of the device delivers its functions.

Each design case in Kritik contains a case-specific structure-behavior-function (SBF) model, where the SBF models guide the adaptation of old designs to meet new functional specifications. The adaptation process result in the design of new devices and the construction of new SBF models that enable an evaluation of the new designs. In the Kritik experiments, we found that the SBF models not only give rise to adaptation strategies for modifying old designs and for evaluating whether the modified design achieves the desired function, but that they also provide a well-grounded vocabulary for indexing the design cases and enabling case retrieval and storage. Here we focus on the adaptation and evaluation tasks only. The SBF vocabulary enables

Kritik to note the differences between the specifications of a desired function and a similar function delivered by a design retrieved from the case memory. These differences between the functional specifications set up adaptation goals. The SBF model of the retrieved design enables a difference diagnosis that generates hypotheses about the structural elements responsible for the functional differences. Since the behaviors in the model capture teleological and compositional knowledge, they help in localizing diagnostic problem solving. The adaptation goals and the diagnostic hypotheses evoke repair plans, which, when instantiated in the context of the current SBF model lead to a revised model. The revised model enables a qualitative simulation to determine whether its behaviors can deliver the desired functions. If this evaluation fails, then the model is further repaired. This leads to device design and model construction by incremental model revision.

The recently developed IDEAL system inherits all the representational and problem-solving capabilities of Kritik. In addition, it expands and extends the theory of adaptive modeling through the introduction of generic (i.e., case-independent) models for transferring design patterns from one design problem to another, where the target problem and the source analog may be in different domains. More specifically, IDEAL contains knowledge of two kinds of generic design patterns: generic teleological mechanisms and generic physical processes [Stroulia and Goel 1992; Bhatta and Goel 1993, 1994]. A generic mechanism specifies a pattern of functional and causal structure, such as feedback, while a generic process captures a pattern of behavioral and causal structure, such as heat flow. The generic patterns are represented as behavior-function (BF) models, where a BF model is a functional and causal abstraction of over case-specific SBF models. When IDEAL succeeds in solving a design problem by constructing a SBF model for the desired device, it uses the new model, together with similar models in its analog memory, to acquire BF models of generic patterns. This kind of abstraction becomes possible because the SBF model provides a functional and causal explanation of the working of the new device.

The generic models provide IDEAL with an enhanced set of adaptation strategies. As in Kritik, IDEAL uses differences between the specifications of a desired function and a similar function delivered by a design retrieved from the analog memory to set up adaptation goals. Also as in Kritik, it uses the SBF model of the retrieved design for difference diagnosis. Depending on

the specifics of the adaptation goal and the diagnostic hypothesis, IDEAL can evoke the repair strategies that involve use of generic models. Such a repair strategy specifies both a generic model represented declaratively in the form of a BF model and a skeletal plan represented procedurally. The repair plan instantiates the generic model in the context of the current SBF model. For example, one repair plan in IDEAL it inserts feedback loops in an electronic circuit by instantiating the corresponding generic teleological mechanism. Since the generic model of feedback was learned in the context of a different problem that may have been in a different domain, this is a process of analogical transfer. Since this kind of analogical transfer is mediated by generic models, and the generic models themselves are acquired through abstraction from case-specific models, we call this model-based analogy.

5. Computational Modeling

We turn now to the issue of computationally modeling S2's reasoning. Our conceptual analysis of the protocol described in Section 3 provides an interpretation of the protocol data as a form of constructive modeling. This provides the modeling constraints for our computational model and the theory of adaptive modeling briefly sketched in Section 4 provides the representational structures and problem-solving constructs for the task. We are presently developing a computer program called ToRQUE (for Theory Reconstruction through Questions, Understanding, and Evaluation) that accepts these modeling constraints and problem-solving languages as starting points. When complete, the program will enable detailed experimentation with, and potential refinement of, our conceptual analysis of S2's reasoning and both the AI and philosophical theories we are merging here. But, even in its current form, ToRQUE helps to make our conceptual analysis more precise and its assumptions more explicit.

Current work on ToRQUE focuses on two elements of S2's reasoning: model revision and model evaluation. First, the issue of model revision: Given that scientists, such as S2, often possess incomplete mental models, i.e., models which are insufficient for answering certain questions, how are they able to construct more complete models which adequately answer those questions? S2 was placed in this position with the spring problem. He immediately realized this fact: "This is the first problem I've encountered -- I note -- where I had some doubt about being able to solve it." (001) S2 was aware that the knowledge needed to answer the question posed to

him is not readily available in his model of the spring, and therefore has some concern about being able to solve the problem. Second is the issue of model evaluation: once a model is retrieved or constructed, how does the scientist evaluate that model? That is, even when a mental model enables the scientist to answer a question, how does he evaluate the answer to the question? For example, S2 attempted to answer the spring question by making use of a model of an analogous system, the flexible rod. Even though he was able to answer the question, he was unsatisfied with his answer: "But then it occurs to me that there is something clearly wrong with that metaphor..." (005) What enabled S2 to know that his answer may not be correct? Model revision and evaluation represent two computationally significant problems that real scientists face, and by developing processes to model these problems, we come closer to modeling the more creative processes of science.

5.1. Model Revision

The theory of adaptive modeling suggests that one of the ways that incomplete models get completed is through adaptation, where the adaptation processes may involve analogical transfer. This is in contrast to other approaches to model completion (e.g. Gentner, 1983) in that the emphasis in our approach is on applying adaptation strategies to satisfy problem-solving goals and constraints, with analogical transfer representing one family of adaptation strategies. The adaptation goals and constraints originate from the overall reasoning goals of the agent (e.g., design, question answering, data modeling), the differences between the target problem and the source analog, and also the diagnosis of these differences. In our computational model of S2's reasoning, adaptation occurs between the initial model provided by the problem (the spring) and a retrieved Model 1 (the flexible rod), and also between this initial model and constructed Models 2-4 (the zigzag spring, the rigid connector spring, and the polygonal coil). This kind of model adaptation raises four important questions: 1) What is the knowledge content of the models? 2) How is a model searched for finding answers to questions? 3) How are the analog models retrieved? And 4) What parts of the analog (if any) provide the necessary constraints? In the following sections we focus primarily on questions 1, 2, and 4. We have put little emphasis on the retrieval issue (3) in developing TORQUE, not because it is unimportant, but because we consider the other issues to be of greater relevance to this analysis.

5.1.1. Model Contents

We begin by addressing the first of these questions. What is the knowledge content of S2's initial model of a spring? At the end of the protocol session when asked if he had done any past thinking about springs S2 replied: "Absolutely not, none of this resembles any thinking I remember doing, in fact I would say the most I ever did with springs was just coefficient of $F=kx$ or $F=kx$ or whatever it - type of stuff, where it doesn't have to be a spring at all, it can be anything elastic and the fact that it coils is irrelevant." (190) We can infer from the protocol that S2 had (or constructed) a generic model of springs. He knew that springs stretch in the sense that they are elastic, i.e., have a restoring force. We can also infer that he considered the spring to be made up of coils. He referred to the individual coils many times, e.g. "that is to say that the distances between the coils would be equal..." (013) and "I'm visualizing a single coil of a spring" (023). Thus the spring is a device having a function and made up of individual components. Also, there are structural, functional and causal relationships between these components. SBF models therefore are a logical choice for representing spring models

In ToRQUE, we have made two modifications to Kritik's SBF models. First, while Kritik's SBF models were case-specific, ToRQUE uses the SBF language to represent more abstract models, e.g., springs in general as opposed to a specific spring. This allows us to adequately capture the generic concept of spring with which S2 works. Second, we have added spatial information to the SBF models. Since Kritik is interested in function ---> structure conceptual design, the representation of structure in its SBF models was limited to topological relationships between components (e.g., connection, containment) that play a functional role in the working of the device. But the structural representation did not specify the spatial form of the components or the device. In the S2 protocol, however, the spatial form plays a much larger role, as he uses several topological transformations in model construction. The connection of components such as coils, for example, is dependent upon their orientation and the continuity of the connection.

Figures 21, 22, and 23 illustrate the SBF model of a spring without the spatial information - we will return to spatial representation a little later. The model captures only the knowledge that we can safely assume S2 had about springs prior to problem solving. There are many inferences

which can be made from this model, but none which establish a relationship between the diameter of the spring and the stretch-function (as asked for in the question posed to S2).

5.1.2. Model Search

This leads us to our second question: How is a model searched for finding answers to questions such as "how does the diameter relate to the stretch function"? Questions in ToRQUE serve to set up initial reasoning goals. The question processing module accesses the particular function in question and can quickly determine if the diameter property effects or is effected by the stretch function. Depending on the question and the completeness of the initial SBF model, the model may explicitly specify the needed information. If this information is not explicitly specified, then ToRQUE attempts to establish paths of relationships between the variables that are specified in the model and that can answer the question. If this too fails to yield an answer to the question, as in S2's case, ToRQUE shifts to an alternative strategy for model adaptation based analogical transfer.

5.1.3. Analog Retrieval

Retrieval of a relevant model from memory involves the construction of a probe based upon the problem-solving context. Kritik uses functional specification of the target problem as the probe while IDEAL can use both functional and structural specifications for this purpose. The protocol on S2's reasoning provides some evidence for both kinds of probes. A flexible rod has a function nearly identical to that of a spring: they both provide a restoring force when an external force is applied. And S2 talks about how "springiness implies flexibility." Thus one might posit that S2 was looking for a device with a similar function, and thus uses the function of the spring as a probe into the analog memory. But one might also postulate that the retrieval is facilitated by a topological adaptation to the spring model, i.e. the stretching of the spring to its limit. When a spring is stretched to its limit it becomes a rod. That the initial analog model was retrieved through this topological adaptation is supported by the protocol since S2 referred to the flexible rod as the "straightened out" (007) spring model. In this case, the structural form of the straightened out spring would serve as a probe to memory. In any case, as we stated previously

we have not yet implemented analog retrieval in ToRQUE; instead the system assumes that the model for the flexible rod is available in its working memory.

5.1.4. Model Adaptation

Given that we now have a source analog (flexible rod), what parts of the analog model should be used to construct a revised model for the target problem (Figure 8)? We break this question into two parts. First, what are the differences between the old model of the target problem (spring) and the initial analog model (flexible rod)? Second, given salient differences, what adaptations should one make to the target model?

5.1.4.1. Difference Diagnosis

Since S2's problem involves a particular function of the spring (stretch-function), and the SBF model for the spring explicitly specifies the behavior that accomplishes this function, the initial difference diagnosis involves only this behavior (see Figure 22) and the corresponding behavior in the SBF model of the flexible rod (see Figure 25). A search of these two behaviors in ToRQUE reveals that the varying slope in the flexible rod is not present in the spring. In trying to answer the question, "why should the fact that the spring is straightened out make any difference," S2 recognized that the spring has constant slope under the dynamical condition and took it to be a salient constraint of the target problem. A second difference, taken at this point to be only potentially salient, is that the flexible rod has only one component, while the spring has many. In the protocol S2 considered the possible salience of the repeating coiled segments in his construction of Models 2 and 3 (Section 4). ToRQUE recognizes this second difference both in the structural and behavioral representations of the spring and the rod. The stretch-behavior of the spring is caused in part by the behaviors of the individual coils (see Figure 23). But the bend-behavior of the rod does not have any corresponding components.

The process ToRQUE uses to analyze these differences is graph-based. Behaviors in SBF models are represented as directed acyclic graphs in which the nodes represent behavioral states and the links represent state transitions. Differences between the models of target problem and the source analog are generated by tracing the behaviors and constructing a difference graph as each state and transition is traversed. When both behaviors have the same specific state (or

transition) then the corresponding difference node contains pointers to the two compared states (or transitions), as well as information about the slot-values in the particular states (or transitions). If the graphs are not isomorphic, then there may be states and transitions in either the source or the target which are not present in the other. The difference graph adds these nodes as well, and marks them as present in one behavior and not the other. Slot names are placed in one of the following categories in the difference node, depending upon their compared values: (1) in-A-not-B, (2) in-B-not-A, (3) A-and-B-type-equal, (4) A-and-B-value-equal, or (5) A-and-B-unequal.

The various types of differences lead to significantly different types of inference. The varying slope and the transition difference described above are differences of type (1) and (2) respectively, which are treated as very significant, and which cause the reasoner to seek a way to eliminate them. Differences in value (4), however, are less important because analogs are expected to have different values. The issue of which features to consider important is greatly assisted by the SBF representations. This is because SBF models specify only those features that play a causal role in the functioning of the device. Surface level features such as color are not functional and therefore eliminated from consideration.

5.1.4.2. Adaptation Strategies

As in Kritik, the differences between the target and the source set up adaptation goals. These goals are accomplished using an array of adaptation strategies. And, as in IDEAL, some adaptation strategies may make use of generic mechanisms. But while IDEAL's generic mechanisms represent functional and causal transformations (e.g., feedback), ToRQUE's generic mechanisms represent topological transformations. Because of this we first describe the spatial representations that they act upon.

5.1.4.3. Spatial Representation

ToRQUE captures the spatial aspects of models in the form of a volume of voxels (volume elements). We use the notion that an object can be represented as a set of cubes. In the graphics literature this is known as a spatial-occupancy enumeration (see Foley et al, 1990; p. 549). In AI it has been utilized by Glasgow & Papadias (1992) as part of their visualization level of representation. This volumetric representation is useful for capturing information which is not

represented descriptively by the model structure. Also, there are well know graphical techniques for transforming volumetric representations, which makes them convenient.

A difficult issue with respect to any spatial representation is how to relate that information with the non-spatial content of the model. In ToRQUE this relationship is maintained by providing (x,y,z) values to the points of connection between components which correspond to points within the volume. A second means of connection is via equations. Geometrically a spring can be represented by a circular helix. The parametric equation $[f(t) = r \cos t \ i + r \sin t \ j + d \ t \ k]$, where r is the radius of the helix and d is the displacement between coils] can be used to form the volumetric coordinates. Both r (actually $0.5 * \text{diameter}$) and d are properties of the spring which are provided by the initial model. So while we can have a generic model of the spring, the volume must be a specific volume determined by particular values of the spring instance. This corresponds with the intuition that visual representations must capture some prototypical or exemplar case of the abstract model.

5.1.4.4. Constructing Intermediate Models

Intermediate models are constructed by applying relevant adaptation strategies to a model. In S2's case, the relevant adaptation strategies make use of generic topological mechanisms. For example, by applying the generic topological transformation mechanism "Transform-3D-to-2D" to the spring model, the result is Model 2, a zigzag spring (Figure 9 a), which S2 says we are to consider "a..2-dimensional spring..It has no third dimension." (023) The generic mechanism when applied to the spring model performs an orthographic projection on the spring geometry to obtain a 2D picture of the spring. This 2D picture can then be analyzed for its particular geometrical features by simple image processing techniques.

Currently we have identified eight GTTMs that S2 appears to use to construct intermediate models. These mechanisms are:

- Reduce-Repeating-Components (behavioral)
- Transform-Closed-Figure (topological)
- Transform-3D-to-2D (topological)
- Transform-Planar-Orientation (topological)
- Transform-Continuous-to-Discrete (topological)
- Transform-Discrete-to-Continuous (topological)
- Transform-Segment-to-Closed-Figure (topological)

- Transform-Closed-Figure-to-Segment (topological)

The inverse of the first was identified by Bhatta (1995) as a Generic Transformation Mechanism (GTM) called cascading. Cascading is an teleological mechanism used in design to increase the effects of a particular component. For example, placing multiple batteries in serial progression will increase the current, while multiple resistors serially connected serve to increase resistance. We claim that when S2 reduced the spring to a single coil (Model 4), he was using a the reduce-repeating-components GTTM to carry out that adaptation. S2 realized that the fundamental behavior of a single coil is the same as that of the whole spring: "I'm visualizing a single coil of a spring... Here's a good idea. It occurs to me that a single coil of a spring wrapped once around is the same as a whole spring" (023) Here S2 noticed that all the components are the same, and therefore might make sense to look at just one of the components. The difference between Reduce-Repeating-Components and Cascading is that the cascading mechanism is used to change the function of the design, while the Reduce-Repeating-Components mechanism is used to simplify the model in question.

5.1.4.5. Topology Transformations

The seven remaining generic transformation mechanisms belong to a new class of adaptation strategies that we have begun to investigate in this work: generic topology transformation mechanisms (GTTMs) GTTMs represent transformations to spatial forms of models. They also represent knowledge of how topology transformations lead to structural and behavioral changes. In considering GTTMs we need to address how topological adaptations lead to structural and behavioral changes.

The first GTTM that we have investigated is the "Transform-Closed-Figures" mechanism. (see Figure 10) S2 first brought up this mechanism at (023) when he asked: "Why does it have to be a coil? Surely you could coil a spring in squares, let's say, and it wouldn't -- it would still behave more or less the same..." Here S2 was calling the circularity into question. He suggested that he could view the coils as squares. We take this as evidence of a GTTM which allows S2 to transform closed figures such as circles into other closed shapes such as polygons.

At this point, however, thinking of square coils led S2 to have a visualization of the zigzag spring rather than following out the consequences of the Transform-Closed-Figures mechanism.

Returning to this mechanism later in the session (117) led him to solve the problem. The structural transformation of a circular component into a hexagon and then to a square involves replacing the coil component with a number of flexible-rod components. Thus, the topological change of transforming a circle into a polygon yields a structural change of replacing the coil components with bending rod components oriented at right angles to one another. Behavioral evaluation of this model leads to the discovery of torque in the coil (see Figure 19).

The second topological GTTM is the "Transform-3D-to-2D" mechanism (see Figure 13). We hypothesize that there is a class of topological mechanisms for transforming the dimensionality of spatial representations. Currently we can only substantiate one member of this class, the "Transform-3D-to-2D" mechanism. This GTTM acts on the spatial occupancy array by taking an orthographic projection of the volume. An orthographic projection is performed by multiplying each point in the volume by a simple transformation matrix. Figure 13 shows that there are 2 possible projections of a spring. The projection from the x or y perspective, and the projection from the z perspective. Once a 2D diagram is obtained, a simple image processing algorithm is needed to identify the line segments and connection points in the figure. The segments are interpreted as flexible rod components within the system, and the connection points as joints. Currently ToRQUE is supplied this information, i.e., it is given that coils when viewed orthographically from the x, y perspective are composed of two segments connected at a joint. Structurally, this topological mechanism replaces each coil component with 2 flexible rod components and a joint component. Since the bendiness of the zigzag spring is all located at the joints, the behavior of this model under the dynamical condition is of the same type as that of flexible rod model and not of the spring.

The "Transform-Segment-to-Closed-Figure" mechanism and its inverse (see Figure 12), "Transform-Closed-Figure-to-Segment" capture the idea that we are able to mentally transform simple line segments into closed figures and vice versa. This topological transformation in the Clement case allows for the structural transformation from a coil component to flexible-rod component and vice versa.

The "Transform-Planar-Orientation" mechanism changes the orientation of a component by 90 or 180 degrees in some direction (see Figure 14). This GTTM is especially useful for modeling the effects of force because force effects components differently depending upon their orientation with respect to the direction of application of the force. In ToRQUE we have captured the "orientation of force" as either: (1) perpendicular to the connection, (2) parallel to the connection, or (3) normal to the connection (see Figure 20).

Topologically, the "Transform-Planar-Orientation" mechanism involves a rotation of the volume through multiplication by a simple rotation matrix (see Foley et al, p.215). For example a simple 90-degree rotation about the x axis is:

$$R_x(90) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 90 & -\sin 90 & 0 \\ 0 & \sin 90 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

If the force remains in its original direction, then structurally the model is adapted by changing the orientation of flow with respect to the components. So, for example, when S2 rotated the bent up rod, and thus the hexagonal coil, from the xy plane to the xz plane (117), we have the orientation of flow change from perpendicular to normal in ToRQUE.

The "Transform-Discrete-to-Continuous" mechanism and its inverse "Transform-Continuous-to-Discrete" are both topological mechanisms. S2 uses these mechanism in several ways: in simulating bending the rod into a series of polygons that would "produce a series of successive approximations to the circle" (117- 118) to "construct a continuous bridge" between the rod and a spring coil; in recognizing that there is not "a lot of difference between the circle [series of rod segments taken to limit] and say a hexagon"; and in distributing the torsion discovered in the polygonal model the circular spring coil (Figure 9 c) In the latter case, e.g., he correctly attributed this mechanism as a limiting case (see Weld, 1986): "What I'm sorta wondering is what happens to the torsion in the limit... what happens to the respect of contribution of torsion and bend in the limit as one keeps making the segments smaller..." (162)

The Transform-Discrete-to-Continuous GTTM adapts the structure of a model by replacing the connected segments of the polygon with one continuous component which contains the necessary differential information. Thus given that the device is made up of multiple equivalent components that perform some discrete amount of a particular function. This

mechanism replaces those components with a single component which distributes that functionality evenly. In S2's case this information is the uniform twisting along the length of the coils.

The Transform-Continuous-to-Discrete GTTM is a mechanism whereby the system can break a component into small pieces. This is done by choosing a size of the infinitesimal and then creating enough infinitesimal pieces to equal the length, area, or volume of the original component. Currently this strategy involves only lines and curves. Thus we can break a circle or line segment into many pieces (see Figure 11). In the Clement example, each of these line segments are interpreted as lengths of wire, which can be viewed as flexible rod components.

5.1.5. Model Revision Summary

In this section we have shown how models can be revised through a process of adaptation involving the use of generic topology transformation mechanisms. We have shown how modified SBF Models capture the necessary content of the models, how a model can be searched when required, how the agent retrieves possible analog models, and finally how structural, behavioral, and topological generic adaptation mechanisms can be used to form intermediate models. Figures 17 and 18 show the progression of models and which adaptation mechanisms were used in the S2 protocol to adapt them. First (a) shows how S2 would uncoil a spring by use of the "Transform-Closed-Figure-to-Segment" mechanism. In (b) the "Reduce-Repeating-Components mechanism" is used to reduce the spring to a single coil. In (c) we see the formation of the zigzag wire, and in (d) is the coiling of the rod into a coil. In (e) is the progression of closed-figure transformations, which lead to the hexagonal coil and the discovery of torque (see Figure 19). Finally, (f) shows the transformations from the square coil back to an adapted model of the circular coil. We claim that this progression of adaptations captures the reasoning which takes place in the protocols, and is a step towards a computational theory of scientific discovery through model revision.

5.2. Model Evaluation

The second major computational issue we address in this paper is how do scientists evaluate the models that they have constructed? First we need to distinguish between two forms

of evaluation: internal and external (see Bhatta 1995, chapter 6). External evaluation involves an experiment carried out in the world, while internal evaluation is a reasoning process which acts on currently held models. The theory of constructive modeling developed in Section 2 hypothesizes that internal evaluation often involves some kind of mental simulation of the model. In the protocol situation S2 was not provided any springs with which to experiment, so we have focused only on internal evaluation. To begin with, we can see that evaluation runs through the whole constructive modeling process -- its goal is to assess the current state of the model and to adapt that model so that it can solve the current problems. As our research develops we will need to address three issues: (1) How does the scientist determine whether a model - constructed or retrieved - contains the content necessary to solve pending problems? (2) How does the scientist decide that a model adequately answers the problem at hand? And (3) how does the scientist determine whether models are internally consistent and coherent? Thus far we have only addressed the first two issues and will discuss them below.

5.2.1. Evaluating Content

Retrieving or constructing an analog model does not guarantee that the analog will have the content necessary for solving the problem. For example, when S2 retrieved the model of the flexible rod, there was no guarantee that the flexible rod would provide an answer to the problem. In order for S2 to solve the problem, he must ask the same or a similar question of the flexible rod as was asked of the spring in the problem formulation. In ToRQUE we reformulate that question as "how does the diameter relate to the stretch function?" A flexible rod, however, does not have a diameter. Thus, computationally one may be tempted to give up on the flexible rod model as a possible analog. But we know that S2 was able to propose a solution to the problem by referring to the flexible rod. Here we claim that past processing of the problem with regard to the spring allows S2 to answer the question using the flexible rod model.

As S2 attempted to relate the diameter of the spring to its stretch function, he first looked for some direct relationship, which in this case is to look for the diameter property within the stretch-function of the spring. When this attempt failed, he looked for a secondary relationship between the diameter and the stretch-function, i.e., is there anything about the rod that can be put in correspondence with the diameter of a spring? And if so, are any of these properties contained

in the stretch-function? We can assume that S2 knew that the diameter is directly related to the coiled length of the spring by a factor of π . The coiled-length, however, is not part of the stretch function either, which is why S2 failed to answer the question. The flexible rod does not have a diameter, but it does have a length, and the coiled length of the spring corresponds with the rod length of the flexible rod. Thus the new question becomes "how does the rod length relate to the stretch function?" Also, because the flexible-rod was retrieved via its function, we can replace the stretch function in the question with "bend function". So the final question for the flexible rod model is "how does the rod-length relate to the bend function?" This question can be answered directly with the model of the flexible rod.

In sum, ToRQUE evaluates intermediate models for their content by attempting to ask the same question of the analog as was asked of the source. If this fails to produce an answer, it adapts the question based on reasoning carried out when attempting to answer the question with the source model. Of course, the solution provided is not guaranteed to be correct with regard to the original model. So we must now turn to second evaluation issue: how to evaluate solutions.

5.2.2. Evaluating Solutions

The first solution obtained by S2 was the flexible-rod - or "straightened spring" - model solution. It was clear to S2 that a flexible rod of greater length will bend more given the same amount of force, yet he is unconvinced that this answer is relevant to the spring case because of salient difference in the constraints in the dynamical case. Thus, the evaluation of the solution is dependent upon what differences the scientist finds to be salient. It is quite possible that the scientist does not view any of the differences as salient, in which case the scientist will claim to have solved the problem. This was the case with many of Clement's other subjects who were satisfied with the rod model. S2, however, did notice a number of differences including the varying slope of the flexible rod and this led to further model construction. We maintain that the construction of intermediate models is a method of evaluation. The intermediate models are constructed to uncover and clarify the differences between the original model and the analog solution. ToRQUE evaluates solutions through a process of establishing the differences between the source and analog models (see section 5.1.4.1) and constructing models which attempt to eliminate those differences.

5.2.4. Model Evaluation Summary

In this section we have shown how the construction of intermediate models is used to evaluate retrieved models and solutions. We have shown how a scientist evaluates the content of a model to determine if it can help solve the problem. We have shown how a scientist evaluates the solutions to models through the construction of intermediate models aimed at eliminating differences. We are just beginning in our research to address how a scientist evaluates a model through simulation and the retrieval of generic physical principles.

5.3. Summary of Process

ToRQUE employs the coarse-grained commitments of constructive modeling along with the design theories of adaptive modeling and model-based analogy, to construct models based on the differences between the source model and a retrieved analog model. This process occurs within a problem-solving context, e.g., when a question is asked of the source, ToRQUE attempts to answer the question by finding relationships within the source model. (see Section 5.1.2) If ToRQUE cannot answer the question directly, it retrieves an analog model. The analog model is retrieved based upon a probe constructed from the problem solving context, e.g., a match between the function of the source model and the analog model. (see Section 5.1.3) A similar question is then asked of the analog model. If the analog model fails to provide an answer, another analog is sought. If it does return an answer, the differences between the source and analog models are assessed to verify that answer. (see Section 5.1.4.1) If the functions of the two models match, ToRQUE assumes that their behaviors will also have some amount of similarity. Thus, in this case, the behavioral differences help to constrain the number of choices for construction, even though there is no guarantee that the spatial and structural differences between the two models will be at all similar. This is evidenced by the spring and the flexible rod, which vary on almost all structural and topological dimensions. The differences between the behaviors then constrain ToRQUE to look at only specific spatial and structural differences based upon their differing behaviors. (Notice that if retrieval is based on a topological similarity rather than a functional similarity, then the construction will follow a different path. The Clement protocol, however, does not initially provide this kind of problem context.) Once the

differences between the models have been established, GTTMs are retrieved based on those differences. These GTTMs are then used to adapt the source model in some way. The process then continues by evaluating the constructed models, and adapting them to construct new intermediate models.

6. General Discussion

As we stated in the introduction to this paper, two independently developed but closely related theories underlie our analysis of creative scientific problem solving of the kind exhibited in Clement's protocol. The first theory is a philosophical theory that posits "constructive modeling" as a productive form of reasoning in conceptual change in science. This theory is based on our prior analyses of reasoning in historical cases of scientific discovery. The second theory is an AI theory of "adaptive modeling" which provides a content account both of mental models of how physical devices work and of problem solving in constructing models for new systems. This theory arises from our earlier work on creative conceptual design. We focus in this section on why we believe the AI theory of adaptive modeling is appropriate for analyzing the kind of reasoning posited by constructive modeling computationally both in general and in terms of our implementation thus far.

A central piece of the philosophical theory is an hypothesis about a form of reasoning that is productive of representational change in science, i.e., constructive modeling. We posited that in the S2 problem-solving session, the reasoning process is of the same type as the constructive modeling performed by historical scientists such as Maxwell. In its present form this model enables a consistent conceptual analysis of S2's reasoning in solving the spring problem as exhibited in the protocol transcript. In the current phase of our research, the conceptual analysis of the S2 protocol provides a bridge between the philosophical and the AI theories. The data provided in the protocol transcript do not "speak for themselves". They could admit of more traditional interpretations than as an instance of constructive modeling. However, we have indicated, as in the historical cases, how constructive modeling provides the best fit. It is the nature of the constructive modeling interpretation of this kind of scientific reasoning that makes our selection of adaptive modeling the appropriate AI theory.

The constructive modeling hypothesis is in the form of a cognitive-historical model that makes high-level, coarse-grained representational and processing commitments that provide a framework for understanding representational change. The AI theory takes the form of a computational model that makes specific representational commitments in the form of SBF models of device comprehension, and specific processing commitments in the form of MBA models of analogical reasoning. The AI theory provides a language for construction of precise and detailed computational models of scientific discovery. The SBF models explicitly represent structures, functions, and behaviors of a device, where the behaviors mediate between the device structure and function, and capture teleological and compositional knowledge. This account also includes generic models, for example, generic topology transformation mechanisms. The latter content account is in the form of problem-solving goals and tasks, strategies and methods, inference and control. Thus, new device models are constructed by adapting known models, and adaptation strategies range from simple and direct revision of a known model to analogical transfer from other device models, where the transfer is mediated by generic models. The two content accounts are closely related: the knowledge content of the models enables the inferences required by problem solving.

Interestingly, the theory of adaptive modeling appears to cover some earlier work on scientific reasoning, specifically Darden's work on the Transgene project [Darden 1991a, 1991b]. She represented early theories of plant genetics as a device model and viewed data-driven theory revision as a kind of device redesign or model revision. The correspondence between our theory of adaptive modeling and Darden's work on Transgene is not accidental. She used Chandrasekaran's functional representation scheme [Sembugamoorthy and Chandrasekaran 1986; Chandrasekaran, Goel and Iwasaki 1993] to represent models. Our SBF models too evolve from the functional representation scheme. Also, Darden's model revision strategies were closely related to Kritik's strategy for device redesign through model revision. The adaptive modeling theory evolves from our earlier work on the Kritik project.

Constructive modeling makes commitments about how scientists use analogy, generic abstraction of various sorts, and visual/spatial reasoning to solve problems. It also holds that knowledge is contained in models. Adaptive modeling is a computational theory that can provide the requisite precision to many of the aspects of constructive modeling. Adaptive modeling

provides a precise account of how models are adapted through the use of generic abstraction, generic strategies, and generic mechanisms. Both constructive modeling and adaptive modeling are incremental strategies for constructing models of physical systems. The SBF models of the computational theory provide a precise means for representing causal and structural knowledge which are clearly necessary from the perspective of constructive modeling. The constraints formulated within the framework of constructive modeling are given precision by the separation into structural, behavioral, and functional categories. So SBF models provide a computationally tractable way of assessing the constraints postulated by constructive modeling.

Our conceptual analysis of S2's reasoning and our computational modeling in ToRQUE seem to suggest that SBF models can capture some key elements in S2's mental models of systems such as springs and flexible rods. For example, there appears to be ample evidence in the protocol that S2 has representations and processes which deal with the structure, behaviors and functions of these systems. This is noteworthy because other AI accounts of device models based on "naive physics" [Hayes 1979] and "qualitative physics" [de Kleer 1984] not only do not represent device functions, but explicitly forbid their representation. Further, these alternative accounts derive the system behaviors by qualitative simulation at problem-solving time. There is little evidence of this kind of simulation-based derivation of system behaviors in S2's protocol. Instead, S2 appears to either already have the behaviors compiled in his device models, as in SBF models, or adapts the behaviors of one system (e.g., flexible rod) to derive the behaviors of another (e.g., the spring), as suggested by the adaptive modeling theory. Thus as in constructive modeling simulation is an evaluative tool rather than a derivational one.

The resemblance between our conceptual analysis of S2's reasoning and the problem solving method in adaptive modeling is quite striking. One of the central issues in model construction through adaptation is the spawning of the adaptation goals: what problem constraints are instrumental in formulating adaptation goals? According to the theory of adaptive modeling, the differences between the models of the target problem and the source analog set up the adaptation goals [Goel 1991a, 1991b]. Much of the adaptation process is driven by these differences. We can see this is also a central feature of S2's reasoning. The difference in the change of slope in the models of the spring and the flexible rod drive his construction of Models

2, 3 and 4. These models are constructed primarily to either reduce the difference in the change in slope or explain it away as inconsequential for the problem at hand.

Model-Based Analogy provides a precise theory of analogical transfer which coheres with the kind of analogical transfer required in constructive modeling. As was mentioned above, the analogical process in MBA is significantly different from other models of analogy. The primary difference is its use of generic abstraction, which is a central process in constructive modeling. An advantage of generic abstraction is that it provides a precise account of what should get transferred from the source to the target. By this we mean, what should be abstracted from the source as relevant to the problem at hand. Other analogical systems such as SME (Gentner, 1993) look for the deepest level of correspondence between source and target. But here the issue is what is generic to both source and target. Another core issue in model construction through adaptation concerns problem-solving strategies: what adaptation methods are suitable for different kinds of adaptation goals? As mentioned earlier, according to the theory of adaptive modeling, the adaptation strategies range from simple revision of a known model to analogical transfer [Stroulia and Goel 1992; Bhatta and Goel 1993]. Further, analogical transfer is mediated by generic models. This too is present in S2's reasoning on the constructive modeling interpretation.

In our conceptual analysis of S2's reasoning, there is also significant use of generic models as postulated by the adaptive modeling theory. In Section 5 we discussed how we started with generic models that captured functional, behavioral and causal structures, and had to complement them with generic mechanisms that capture topological structures and enable topological transformations. Similarly, we had to extend the SBF models with spatial representations in the form of voxels. This is primarily because the classes of conceptual design problems we had studied earlier did not require explicit spatial representations and topological transformations. In general, spatial representations and topological transformations appear to play an important role in creative problem solving both in the natural sciences and in engineering design.

We have shown how adaptive modeling through SBF models and MBA theory can provide a plausible language for a constructivist account of scientific reasoning. Of course, theories of scientific model construction represented by Transgene and ToRQUE always can be implemented in other, more general, AI languages such as that of problem spaces and production

rules. Indeed, Bacon uses precisely this language for recognizing algebraic patterns in scientific data. But irrespective of the language of implementation, first a content account is needed. Once a content account is available, it can be implemented in a variety of languages. The theory of adaptive modeling provides such a content account. For example, it provides the kinds of problem spaces that will need to be set up, the kinds of the domain and control knowledge that will need to be represented in any implementation. Therefore, by our analysis adaptive modeling is clearly the best fit.

Acknowledgments

This work has been supported by the Office of Naval Research (research contract N00014-92-J-1234) and the National Science Foundation (research grant IRI-92-10925). Also, the analysis in sections 2 and 3 was developed by the first author in collaboration with James G. Greeno. We thank John Clement for allowing us to use the drawings and transcripts of the S2 protocol.

References

- Bhatta, Sambasiva R. (1995) Ph.D. Dissertation, *Model-Based Analogy in Innovative Device Design*, College of Computing, Georgia Institute of Technology
- Bhatta, Sambasiva R. & Goel, Ashok K. (1993) Learning Generic Mechanisms from Experiences for Analogical Reasoning. In *Proc. Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, Colorado, July 1993, pp. 237-242, Hillsdale, NJ: Lawrence Erlbaum.
- Bhatta, Sambasiva R. & Goel, Ashok K. (1994) Discovery of Physical Principles from Design Experiences. In *International Journal of AI EDAM* special issue on "Machine Learning in Design".
- Bhatta, Sambasiva R. & Goel, Ashok K. (1994) Model-Based Discovery of Physical Principles from Design Experiences. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Special Issue on Machine Learning in Design, 8(2):113-123, May 1994.
- Chalmers, A.F. (1986) The Heuristic Role of Maxwell's Mechanical Model of Electromagnetic Phenomena. *Studies in the History and Philosophy of Science*, vol. 17, pp. 415-27.
- Chandrasekaran, B. (1986) Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design, IEEE, Fall 1986
- Chandrasekaran, B. & Goel, Ashok K. & Yumi Iwasaki. (1993) Functional Representation as a Basis for Design Rationale. *IEEE Computer*, 26(1):48-56, January 1993.

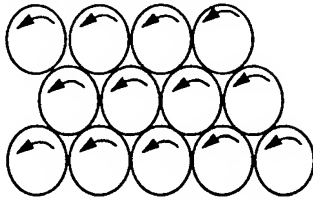
- Chi, Michelene T. H., Feltovich, P.J., & Glaser, R. (1981) Categorization and Representation of Physics Problems by Experts and Novices, *Cognitive Science*, 5, pp. 121-152,
- Clement, John (1989) Learning via Model Construction and Criticism: Protocol Evidence on Sources of Creativity in Science, In *Handbook of Creativity: Assessment, Theory and Research*, Glover, G., Ronning, R., & Reynolds, C. (Eds.), chapter 20, pp. 341-381. New York, NY: Plenum.
- Darden, Lindley (1991a) *Theory Change in Science: Strategies from Mendelian Genetics*. New York: Oxford University Press.
- Darden, Lindley (1991b) Anomaly Driven Redesign of a Scientific Theory: The TRANSGENE.2 Experiments, Technical Report, Ohio State University.
- de Kleer, J. (1984) How Circuits Work. *Artificial Intelligence*, 24:205-280, 1984.
- Duhem, P. (1902) *Les theories electriques de J. Clerk Maxwell: Etude historique et critique*. Paris: A. Hermann & Cie.
- Duhem, P. (1914) *The Aim and Structure of Physical Theory*. New York: Atheneum, 1962.
- Feyerabend, Paul K. (1962): "Explanation, reduction and empiricism" in *Scientific Explanation, Space, and Time*, H. Feigl and G. Maxwell, eds., Minnesota Studies in the Philosophy of Science 3 (Minneapolis: University of Minnesota Press). pp. 28-97.
- Foley, J.D., & van Dam, A. & Feiner, S.K. & Hughes, J.F. (1990) *Computer Graphics: Principles and Practice*, Second Edition. Addison-Wesley.
- Gentner, D. (1983) Structure-Mapping: A Theoretical Framework for Analogy, 1983. Reprinted in *Readings in Cognitive Science*, Collins & Smith (eds.), section 3.2, pp. 303.
- Gentner, D. and Gentner D.R. (1983) "Flowing Waters and Teeming Crowds: Mental Models of Electricity." In *Mental Models*, (eds.) D. Gentner and A. Stevens. (Hillsdale, NJ: Lawrence Erlbaum). pp. 99-133.
- Gick, M.L. & Holyoak, K.J. (1980) "Analogical Problem Solving." *Cognitive Psychology*, 12:306-355.
- Giere, R.N. (1988) *Explaining Science: A Cognitive Approach*. Chicago: University of Chicago Press
- Glasgow, Janice & Papadias, Dimitri (1992) Computational Imagery, *Cognitive Science*, 16, pp. 355-394, 1992.
- Goel, Ashok K. (1992) Representation of Design Functions in Experience-Based Design. In *Intelligent Computer Aided Design*, D. Brown, M. Waldron, and H. Yoshikawa (editors), pp. 283-308, Amsterdam, Netherlands: North-Holland, 1992.
- Goel, Ashok, K. & Chandrasekaran, B. (1992) Case-Based Design: A Task Analysis. In *Artificial Intelligence Approaches to Engineering Design*, Volume II: *Innovative Design*, C. Tong and D. Sriram (eds.) pp. 165-184, San Diego: Academic Press, 1992.

- Goel, Ashok, K. & Chandrasekaran, B. (1989) Functional Representation of Designs and Redesign Problem Solving. *Proc. Eleventh International Joint Conference on Artificial Intelligence*, pp. 1388-1394.
- Goel, Ashok, K. (1989) Ph.D. Dissertation, Integrating Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving,, Department of Computer and Information Science, The Ohio State University, 1989.
- Goel, Ashok, K. (1991a) A model-based approach to case adaptation. In *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society*, p. 143-148, Chicago.
- Goel, Ashok, K. (1991b) Model revision: A theory of incremental model learning. In *Proc. of the Eighth International Conference on Machine Learning*, pages 605-609, Chicago.
- Gorman & Carlson (1990) "Interpreting Invention as a Cognitive Process: the Case of Alexander Graham Bell, Thomas Edison, and the Telephone." *Science, Technology, and Human Values* 15:131-164.
- Hayes, Patrick (1979) Naive Physics Manifesto. In *Expert Systems in the Microelectronics Age*, D. Michie (editor), pp. 242-270. Edinburgh, UK: Edinburgh University Press, 1979.
- Hegarty, M. (1992). Mental animation: inferring motion from static displays of mechanical systems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 18(5), 1084--1102.
- Heimann, P.M. (1970) Maxwell and the Modes of Consistent Representation. *Archive for the History of the Exact Sciences*, vol. 6, pp. 171-213.
- Holmes, F. L. (1990) "Argument and Narrative in Scientific Writing." In *The Paper Laboratory: Textual Strategies, Literary Genres, and Disciplinary Traditions in the History of Science*, (ed.) P. Dear.
- Kuhn, T.S. (1962) *The Structure of Scientific Revolutions* International Encyclopedia of Unified Science, vol. 2, no 2. (Chicago: University of Chicago Press), 2nd edn. 1970.
- Langley, Pat & Simon, Herbert A. & Bradshaw, Gary L. (1987) Heuristics for Empirical Discovery, in *Computational Models of Learning*, Ed. L. Bolc, Springer-Verlag.
- Maxwell, James Clerk (1861-2): "On Physical Lines of Force," in *Scientific Papers*, vol. 1, pp. 451-513.
- Nersessian, Nancy J. (1989) Scientific Discovery and Commensurability of Meaning, In *Imre Lakatos and Theories of Scientific Change*, eds. K.Gavroglu, Y. Goudaroulis, P. Nicolacopoulos. pp. 323-334. Princeton, NJ: Kluwer Academic Publishers
- Nersessian, Nancy J. (1992) How Do Scientists Think? Capturing the Dynamics of Conceptual Change in Science, In *Cognitive Models of Science*, ed. R.N. Giere. pp. 3-44. Minneapolis, MN: University of Minnesota Press
- Nersessian, Nancy J. (1993) Opening the Black Box: Cognitive Science and History of Science, *Princeton University: Cognitive Science Laboratory Report #53*, January
- Nersessian, Nancy J. (1995) Constructive Modeling in Creating Scientific Understanding, *Science & Education*, 4: 203-226.

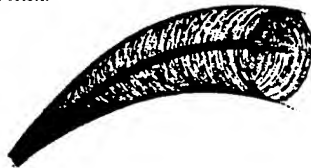
- Nersessian, Nancy J. (in press) Abstraction via Generic Modeling in Concept Formation in Science. In *Idealization in Science*, M.R. Jones & N. Cartwright, eds. (Rodopi)
- Nersessian, Nancy J., (1984) *Faraday to Einstein: Constructing Meaning in Scientific Theories*. Dordrecht: Martinus Nijhoff
- Pierce, C.S. (1931-1958) *Collected Papers*, 8 vols, C. Harstone, P. Weiss & A. Burks (Eds.), Cambridge, MA, Harvard University Press.
- Polya, G. (1954) *Induction and Analogy in Mathematics*, Vol. 1, Princeton University, Princeton.
- Pylyshyn, Z. (1981) "Imagery and Artificial Intelligence," in *Readings in Philosophy of Psychology*, vol. 2, ed. by Ned Block, Cambridge, Mass.: Harvard University Press.
- Sembugamoorthy, V. & Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Riesbeck, editors, *Experience, Memory and Reasoning*, pages 47-53. Lawrence Erlbaum, Hillsdale, N.J.
- Shapere, Dudley (1984) *Reason and the Search for Knowledge*, (Dordrecht: Kluwer Academic Publishers).
- Simon, Herbert (1981) *Sciences of the Artificial* (second edition), MIT Press.
- Stroulia, E. & Goel, A. (1992) "Generic Teleological Mechanisms and their Use in Case Adaptation", In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, 319-324, Lawrence Erlbaum, Hillsdale, N.J.
- Thagard, P. & Gochfeld, D. & Hardy, S. (1992) Visual analogical mapping. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Hillsdale, Erlbaum, 522-527.
- Tweeney, Ryan D. & Gooding, D.C. in process, "Qualitative Skills in Quantitative Thinking: Faraday as a Mathematical Philosopher", manuscript.
- Valdes-Perez, Raul E. (1995) Some Recent Human/Computer Discoveries in Science and What Accounts for Them, *AI Magazine*, vol. 16, no. 3, Fall 1995, pp. 37-44.
- Weld, Daniel (1986) The Use of Aggregation in Causal Simulation, *Artificial Intelligence*, Vol. 30, no. 1, October, 1986.

Maxwell Figures

a) A cross section of the initial vortex medium



b) A single vortex.



c) Maxwell's drawing of idle-wheel particles within the vortex medium

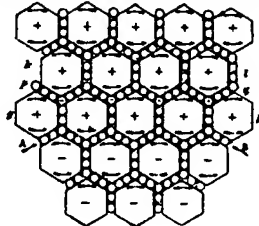


Figure 1

Maxwell's Generic Models

Modes of
Connection:

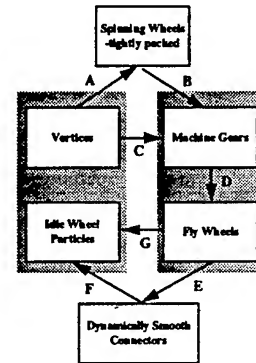


Figure 2

Polya - Abstraction via Generic Modeling

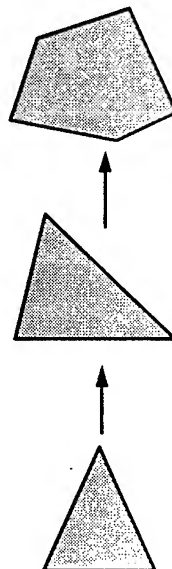


Figure 3

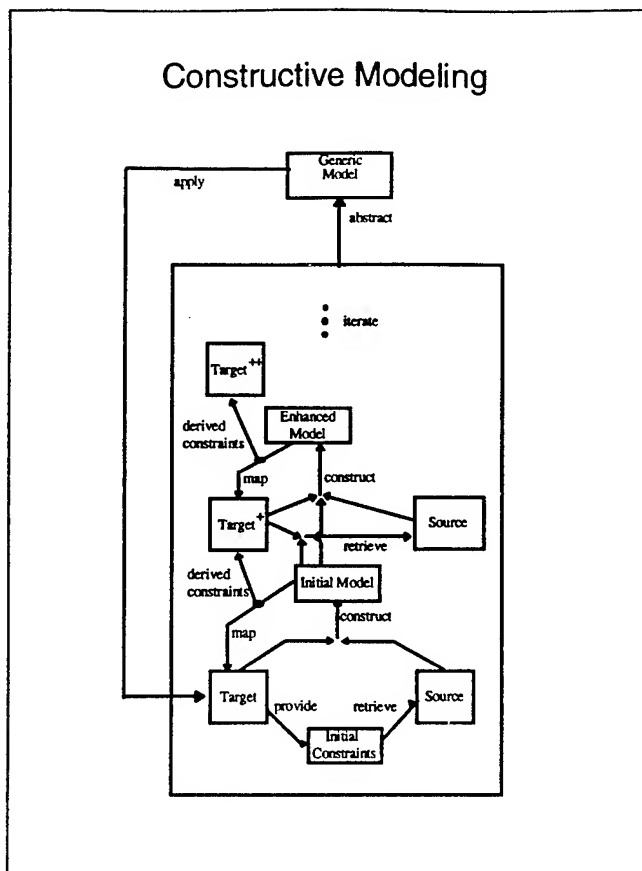


Figure 4

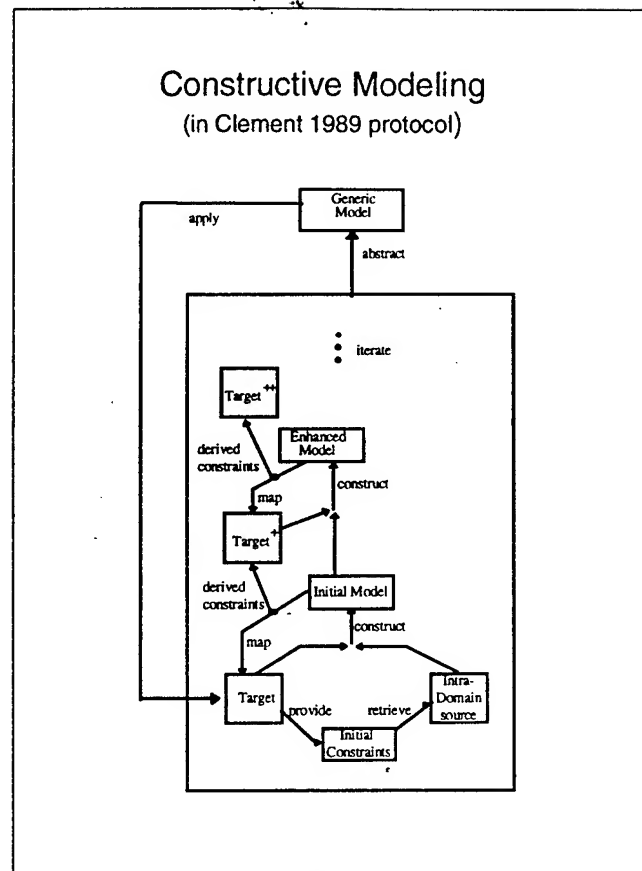


Figure 5

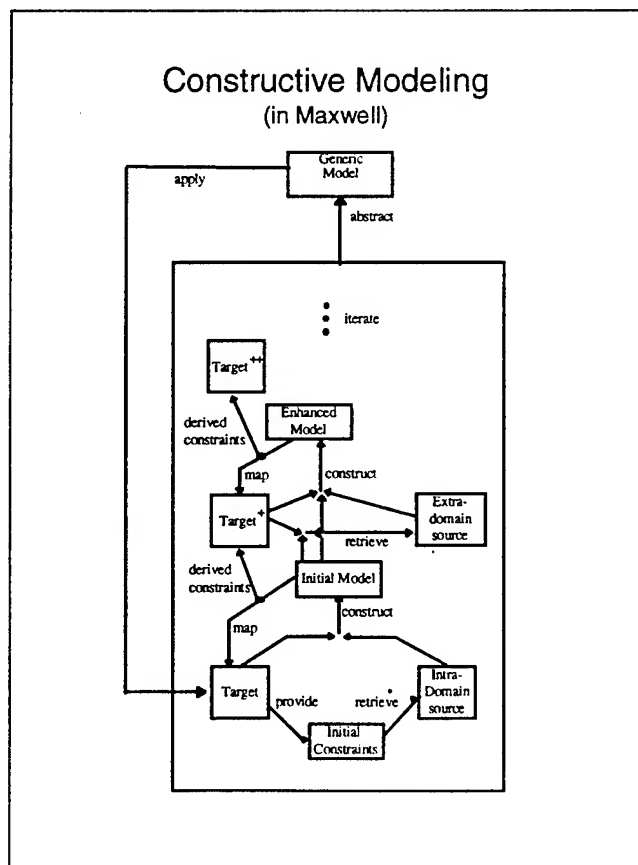


Figure 6

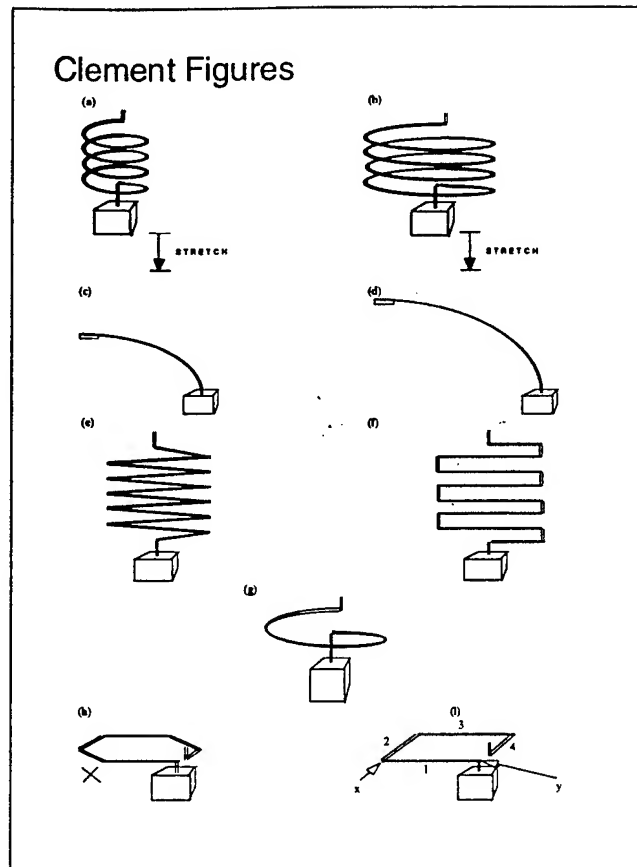


Figure 7

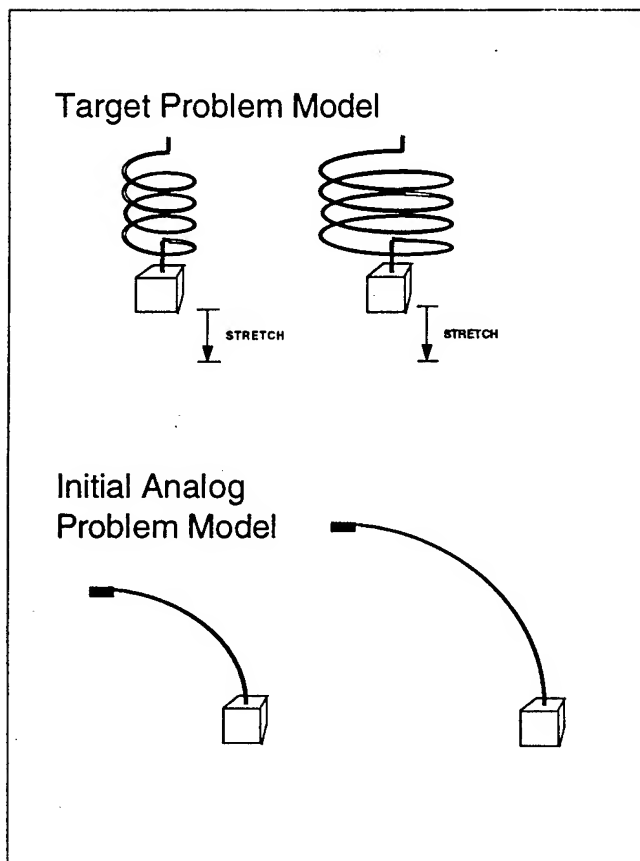


Figure 8

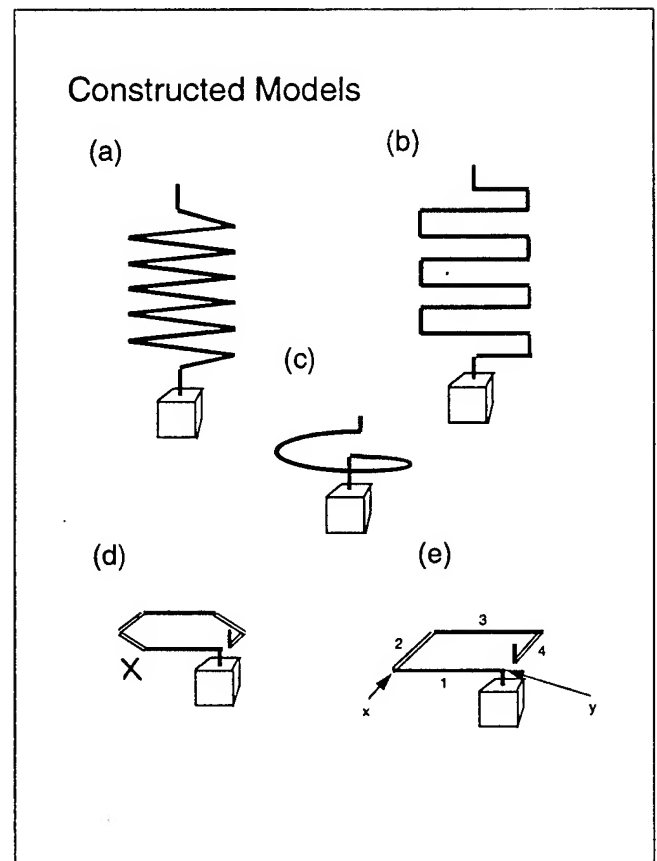


Figure 9

Transform-Closed-Figure (possible transforms)

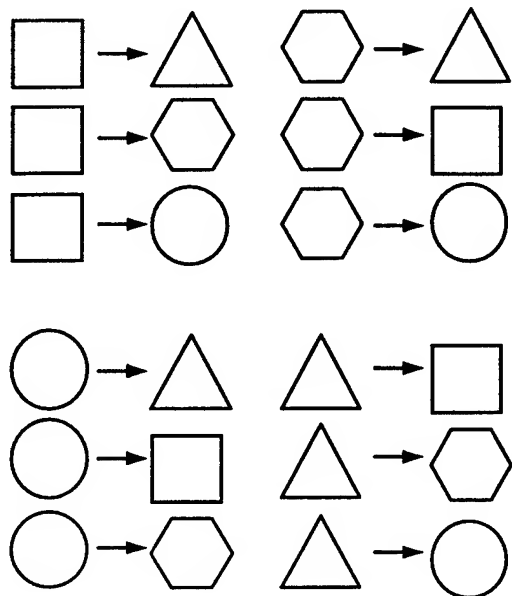
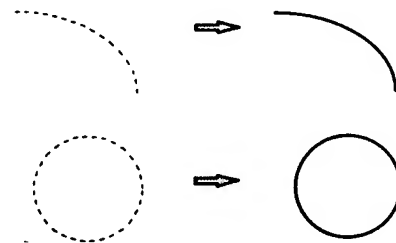


Figure 10

Transform-Discrete-to-Continuous



Transform--Continuous-to-Discrete

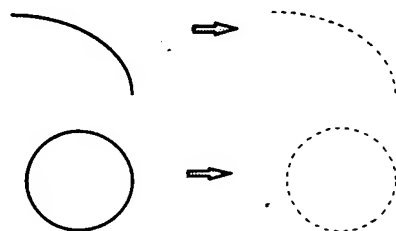
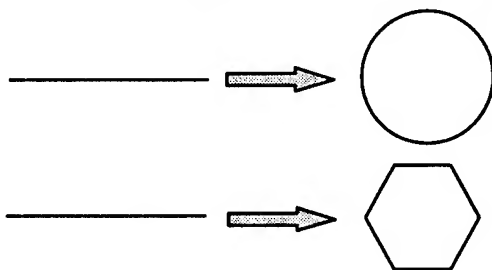


Figure 11

Transform-Segment-to-Closed-Figure



Transform-Closed-Figure-to-Segment

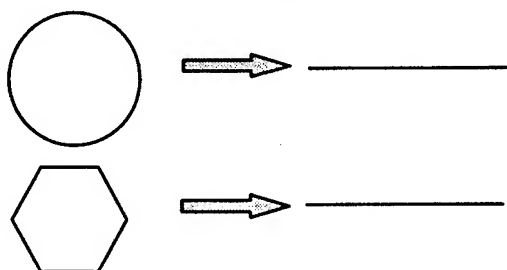


Figure 12

Transform-3D-to-2D

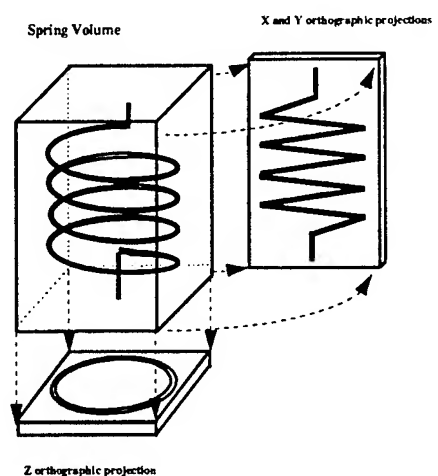


Figure 13

Transform-Planar-Orientation

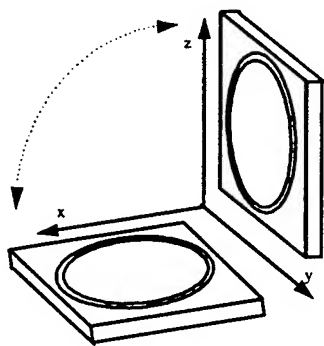


Figure 14

Rigid versus Flexible Joints

Two Behaviors of Constructed Model (a).

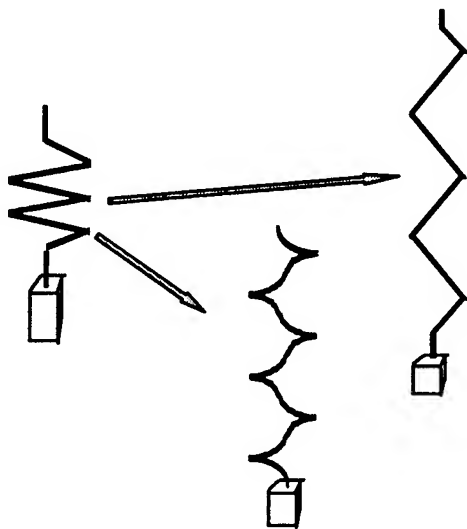


Figure 15

Progression of Models

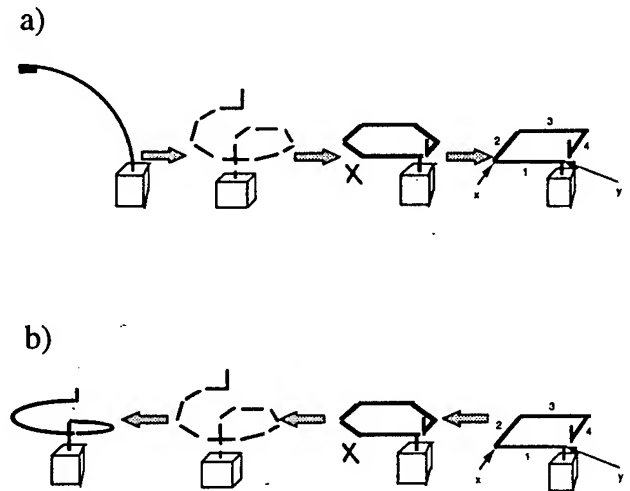


Figure 16

Sequence of Generic Adaptations

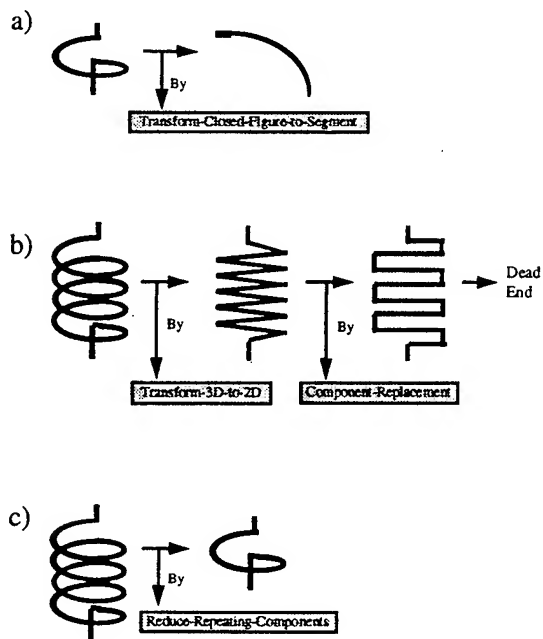


Figure 17

Sequence of Generic Adaptations (continued)

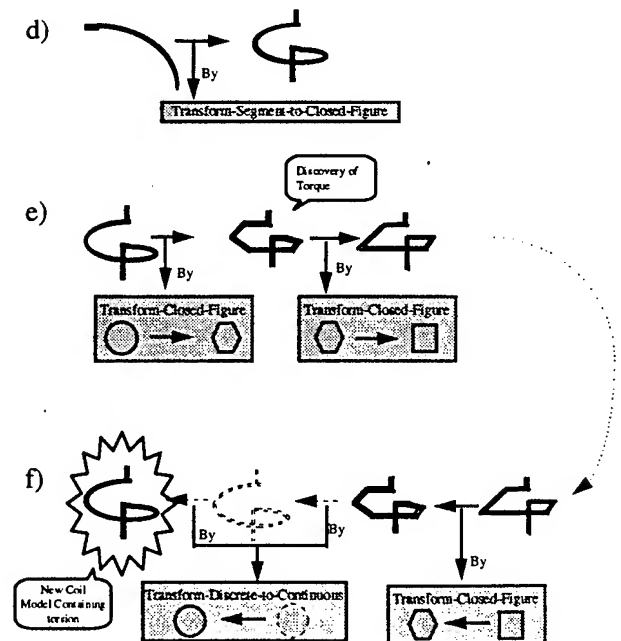
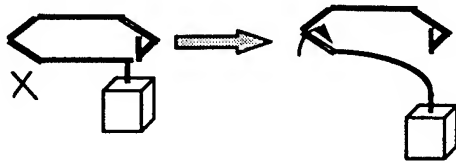


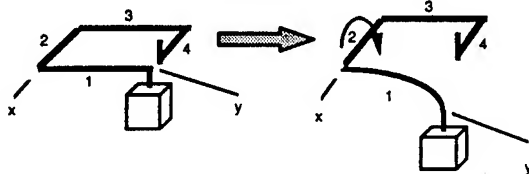
Figure 18

The Discovery of Torque

As recognized through the behavior of constructed model (d).



As exaggerated through the behavior of constructed model (e).



As force is applied at y not only does rod 1 bend, rod 2 twists.
This is torque.

Figure 19

Ontology of Connections

		Flow Orientation		
		Parallel	Perpendicular	Normal
Connection Type	Continuous			
	Noncontinuous	N/A		

♦ = connection between components

Figure 20

SBF Model of a Spring:

Structure

Components: coil₁, coil₂, ..., coil_n

Connections: continuous & serial & normal: e₁|coil₁|e₂, e₂|coil₂|e₃, e₃|coil₃|e₄, ..., e_n|coil_n|e_{n+1}

Properties:

number-of-coils:	n
material:	m
unstretched-length:	usl
stretched-length:	sl
unstretched-slope	uss = usl/cl
stretched-slope:	ss = sl/cl
coiled-length:	cl
constant-of-prop:	k
diameter:	d
tension:	t

Components: coil_i (For all i = 1,...,n)

ends:	e _{coili} , e _{coili+1}
material:	m _{coili} = m
unstretched-length:	usl _{coili} = usl/n
stretched-length:	sl _{coili} = sl/n
unstretched-slope	uss _{coili} = uss
stretched-slope	ss _{coili} = ss
coiled-length:	cl _{coili} = cl/n
constant of prop.:	k _{coili} = k/n
diameter:	d _{coili} = d
tension:	t _{coili} = t/n

Functional abstraction:

Given: coil unstretched
length = usl_{coili}
slope = uss_{coili}
tension = 0

Makes: coil stretched
length = sl_{coili}
slope = ss_{coili}
tension = T_{coili}

Function₁ (Stretch-Function)

Given: spring (E₁, E₂) unstretched
length = usl
slope = uss
tension = 0

Makes: spring (E₁, E₂) stretched
length = sl
slope = ss
tension = T

By: Behavior **STRETCH-BEHAVIOR**
Stimulus: Force F on E₂

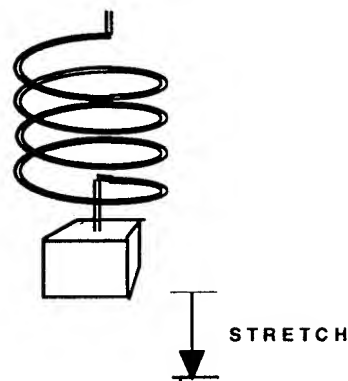


Figure 21

Behavior STRETCH-BEHAVIOR of Spring

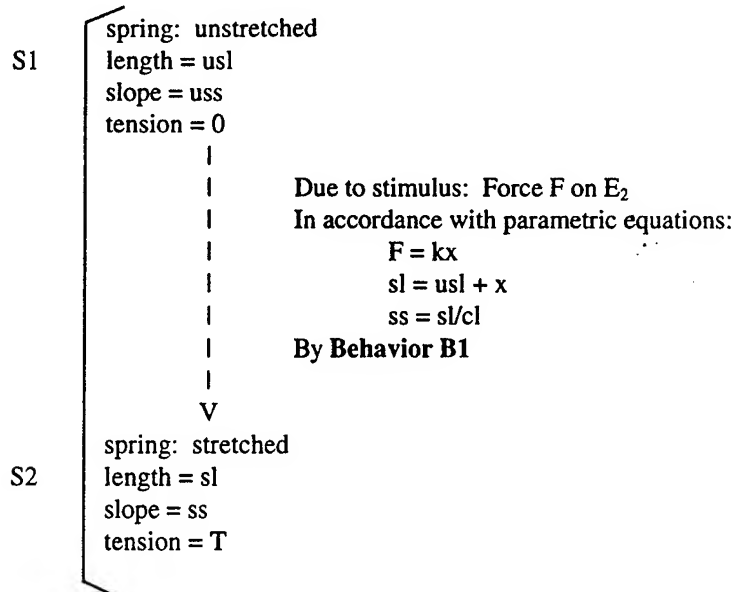


Figure 22

Behavior B1

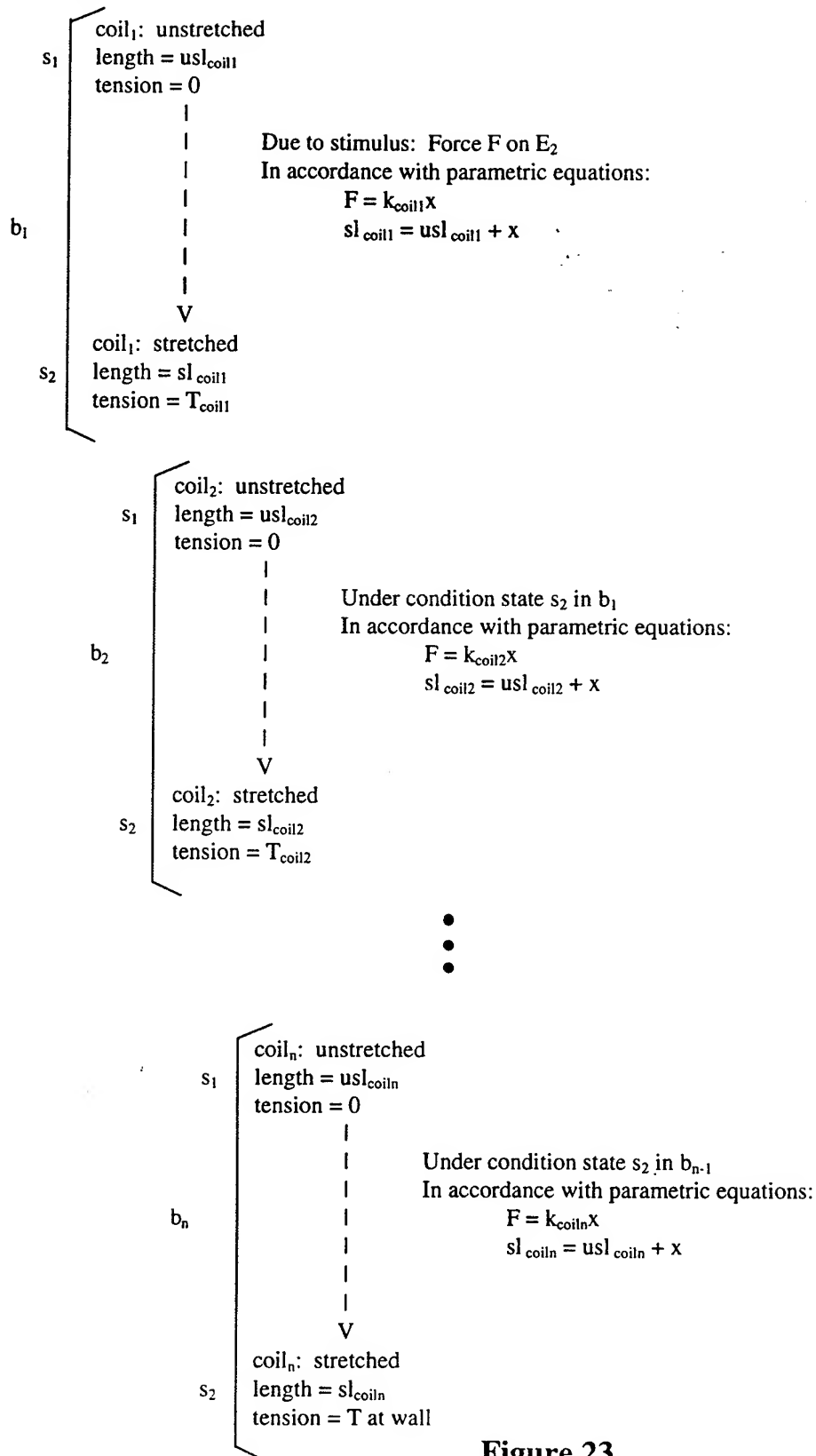


Figure 23

SBF Model for a Flexible Rod

Structure

Components: strip

Connections: serial & perpendicular: E_1 | strip | E_2

Properties:

material:	m
bend-amount:	ba
unbent-slope	ubs
bent-slope:	bs
derivative-of-slope	dslope
constant-of-prop:	k
const-of-length	kl
tension:	t

Functional abstraction:

Given: strip straight
 bend = 0
 slope = ubs
 tension = 0

Makes: strip bent
 bend = ba
 slope = bs
 tension = T

Function₁ = Bend-Function

Given: flexible-rod (E_1, E_2) straight
 bend = 0
 slope = ubs
 dslope =
 tension = 0

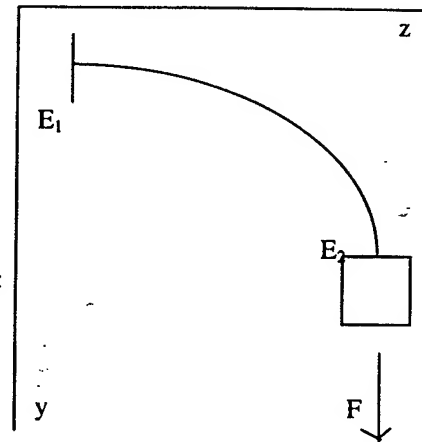
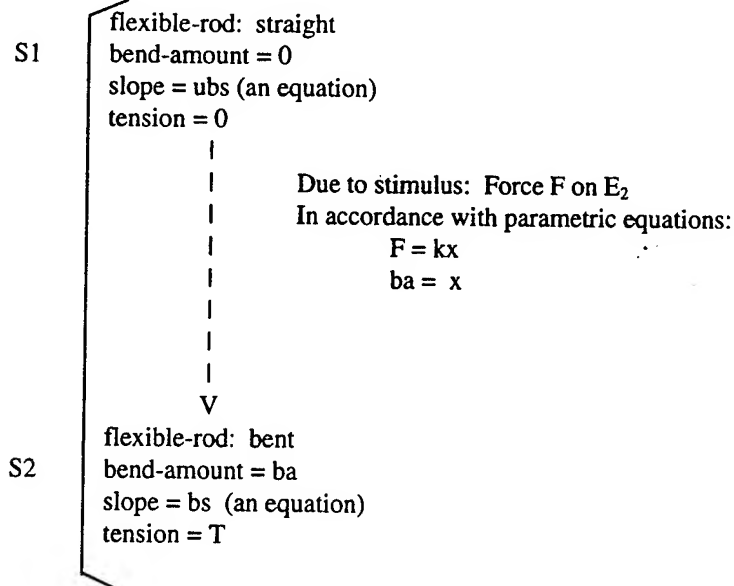
Makes: flexible-rod (E_1, E_2) bent
 bend = ba
 slope = bs
 tension = T

By: Behavior **BEND-BEHAVIOR**

Stimulus: Force F on E_2

Figure 24

Behavior BEND-BEHAVIOR of Flexible Rod



The slope both before and after is equal to an equation because it is varying.
 This equation is the following:

Given that $A < B$:

before: $y = A z^2 + c$ $dz/dy = 2A z$
 after: $y = B z^2 + c$ $dz/dy = 2B z$

Figure 25

Opportunistic Reasoning: A Design Perspective

Marin D. Simina

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
marin@cc.gatech.edu

Janet L. Kolodner

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
jlk@cc.gatech.edu

Abstract

An essential component of opportunistic behavior is *opportunity recognition*, the recognition of those conditions that facilitate the pursuit of some suspended goal. Opportunity recognition is a special case of situation assessment, the process of sizing up a novel situation. The ability to recognize opportunities for reinstating suspended problem contexts (one way in which goals manifest themselves in design) is crucial to creative design. In order to deal with real world opportunity recognition, we attribute limited inferential power to relevant suspended goals. We propose that goals suspended in the working memory monitor the internal (hidden) representations of the currently recognized objects. A suspended goal is satisfied when the current internal representation and a suspended goal "match". We propose a computational model for working memory and we compare it with other relevant theories of opportunistic planning. This working memory model is implemented as part of our IMPROVISER system.

Introduction

During a mechanical engineering project a group of students were asked to design and implement a mechanical device for the quick and safe transportation of a fragile cargo (some eggs). The students went to a Home Depot (a hardware store), where they started by choosing springs for the launching component. During the design process they made the following observations:

Andy: ... hey, when I compress the spring it bends; this weakens the force of the springs ...

Mary (wrapping her hand around the spring): ... yes, we have to enclose it in a tube ...

Bill: ... the tube should be collapsible, otherwise the spring cannot be compressed ...

The students began proposing mechanisms that fit this description. One of them suggested a telescope, but it was rejected by the group because it was expected to be costly and it did not fit in the available budget. Another student proposed a collapsible camping tube, which unfortunately has a wrong shape. The designers were unable to think of where in the store they might look for a useful collapsible tube, so they moved to another part of their problem. They started thinking about load protection. Since sponges are a good way to provide cushioning, they decided to go to the store's bathroom section. During the search for sponges, one of the students saw a toilet paper holder and exclaimed:

Mary: Look! A collapsible tube!

The whole group agreed that the toilet paper holder fulfilled the requirements of their previously suspended problem.

The above example illustrates a rather mundane, but common, experience in doing design. The students started by structuring the initial problem (launching, cushioning) and then they tried to elaborate the subcomponents, one at a time. When they were stuck with one subproblem, they suspended it, and they approached another related subproblem. When they saw the toilet paper holder, however, they recognized that an opportunity to address the suspended subproblem had presented itself in the environment.

What processes are responsible for recognizing such opportunities? How can a cognitive architecture handle this kind of processing? What constraints are there, if any, on the workings of these processes? We are studying these problems in the context of developing a cognitive model for creative design. Our computer program, IMPROVISER (Wills & Kolodner 1994b), was extended in order to help us answer the above questions.

Our exploration of creative design (Kolodner & Wills 1993a) suggests that the conceptual phase, in which the problem is framed, plays a key role in designing. In this phase, which is interspersed throughout the design process, the problem situation is assessed and the given problem is reformulated and restructured. While one can organize the subgoals involved in conceptual design in a hierarchical structure, the processing of these subgoals seems far more unstructured. Designers often begin by proposing a shallow hierarchical set of subgoals as they initially formulate the way they will solve a problem (e.g., the artifact we are designing has these *n* parts or mechanisms; we need to design each one). They continue by addressing each of the subgoals, one at a time. It is here where the organized reasoning breaks down. When the designer fails to solve one subproblem, he/she seems to suspend it and approach another related subproblem (as in the example above). Sometimes the next subproblem is simply a not-yet-considered sibling subgoal (as when the student designers moved from designing their spring launch mechanism to the cushioning for the eggs); sometimes the opportunity to go back to a suspended subgoal is recognized (as when the toilet paper holder was seen).

When we consider the incremental and recursive nature of this reasoning process, we can identify one way of recognizing that a previously-suspended subgoal might be successfully addressed. During consideration of a new subproblem, the designer has to consider interactions with related subproblems, some of which have been suspended previously. This may

provide a fresh view of the suspended problem and a new way to redescribe it. Redescription or new insights about a subproblem gained during reasoning trigger the goal scheduler to unblock the suspended subproblem, allowing already-known solutions to be recalled or new means of solving it to be recognized. This means of unblocking a suspended goal is completely under the control of the reasoner, which knows which subproblems have been part of its most recent reasoning.

But recognizing in the toilet paper holder the opportunity to address a suspended goal requires additional mechanisms that scan the environment and recognize when the environment is providing new insights into suspended goals. If the number of suspended goals, the complexity of the environment, or the amount of newness in the environment is high, such a mechanism could easily be overwhelmed. The mechanisms that provide this capability must be able to deal with such complexity.

Opportunity Recognition

The Problem

The prerequisite for opportunistic behavior is the existence of *suspended goals* (problems), goals that cannot be pursued in the current context and are postponed.

An essential component of opportunistic behavior is *opportunity recognition*, recognition of those conditions that facilitate the pursuit of some suspended goal. But opportunities seem to appear when they are not expected. The student designers, for example, had not previously thought about a toilet paper holder functioning as a collapsible tube. Recognizing the opportunity meant both noticing the toilet paper holder and recognizing that its mechanism (which is hidden) included a collapsible tube. More than a simple matching mechanism is needed.

Birnbaum (1986) suggests two central problems that must be addressed by a theory of opportunistic behavior: (1) how to detect opportunities and (2) how to "activate" the goals to which they pertain. An important issue here is identifying how much and what kind of processing is required in order to recognize the presence of the features that constitute an opportunity.

A Critical Review

Hayes-Roth & Hayes-Roth (1979) proposed the first significant cognitive model of opportunistic behavior. Their model of opportunistic planning was inspired by protocols of subjects planning a hypothetical day's errands. But they were most concerned with planning methods and gave little attention to recognition processes. In fact, the experimenter always mentioned opportunities to the subjects when they overlooked them, and the subjects never tried their plans in the real world, so they never really dealt with genuine opportunities and the problem of recognizing them.

Birnbaum (1986) gave more attention to recognition issues. He proposed the *mental notes* model, in which whenever a goal cannot be immediately satisfied, it is indexed in terms of the unmet preconditions that prevented its satisfaction. However, as he points out, if the goal is indexed too specifically, then there will be many cases in which it will not be recalled even though an opportunity for its satisfaction is present, and if the goal is indexed in terms of more abstract features, we

cannot assume that the agent will automatically generate the abstract description that will activate the goal.

In order to solve the above dilemma within the framework of the mental notes model, Birnbaum suggests¹ spending some effort, when the goal is formed, to determine the range of situations in which it might easily be satisfied – for example, by constructing several incomplete plans for the goal in order to identify the relevant preconditions – and then indexing the goal in terms of the features that might arise in such situations. Birnbaum & Collins (1984) also suggest an active goal framework, where all the goals have the ability to examine the current situation and to initiate inference to test their own relevance.

Patalano, Seifert and Hammond (1992) criticize the use of active goals proposed by Birnbaum & Collins, claiming that this approach to opportunistic behavior is an unlikely explanation of human cognitive processes because of its computational demands. However, Patalano, Seifert and Hammond do pick up on Birnbaum's indexing scheme, calling it *predictive encoding*. Predictive encoding stresses the importance of encoding blocked goals in memory in such a way that they will be recalled by conditions favorable for their solutions. Their experimental results show evidence of this process.

However, the predictive encoding hypothesis seems incomplete, because it does not enable a cognitive agent to recognize opportunities other than those which it is able to anticipate. In particular, it does not enable an agent to recognize novel opportunities, which by their very nature, cannot be easily anticipated. Recognition of the toilet paper holder as a collapsible tube, for example, is novel in that this is not the way a toilet paper holder is generally thought of. Similar issues caused Birnbaum & Collins (1984) to conclude that if an opportunity is to be detected at all, inferential resources must be allocated to the goal recognition task.

Ram & Hunter (1992) suggested a balance between backward chaining at the time of goal suspension and forward chaining at the time of opportunity recognition. In AQUA, a set of utility metrics have been proposed in order to make a tradeoff between predictive encoding and active goals. Unfortunately, these utility metrics are very specific to story understanding.

This suggests that we need active goals in order to recognize novel opportunities, but we need to control their power and number to make them computationally feasible. We need predictive encoding, but we also need more powerful inferential capabilities. We hope that an analysis of the example presented previously can provide insight in formulating a mechanism with these properties.

A Possible Solution

Why did the students fail to remember the toilet paper holder when they were trying to decide where they might find a collapsible tube, and what allowed them to recognize it as appropriate when they saw it?

One possible reason why the toilet paper holder was not recalled and considered while thinking about collapsible tubes is that the *probe* that had been constructed (i.e., the item description used for remembering) was incompletely specified. Consequently, they retrieved items that fulfilled primary but

¹Birnbaum credits Dehn (1989) with this idea.

not secondary characteristics of the probe (e.g., a telescope costs too much and a camping cup has a wrong shape). After every retrieval and evaluation of a new device, the probe was respecified, taking into consideration the initially ignored constraints (e.g., we want something like a telescope, but cheaper). This process was suspended, however, before the toilet paper holder was recalled.

But why was the probe inadequate for retrieving such a common object as a toilet paper holder? Our explanation is that the toilet paper holder is routinely associated with what its purpose is in the bathroom (holding toilet paper rolls) rather than with how this function is achieved (by means of a collapsible tube with a spring inside). It is not a particularly interesting device, and even though we see it every day, most of it is hidden by the roll of paper. Research shows that it is quite difficult to overcome such *functional fixedness* (Mayer 1970), which associates everyday objects with their obvious function (holding a paper roll in the case of the toilet paper holder). Routinely, we ignore other *potential* uses that can be derived from the structure and behavior of such everyday objects. Once we have specified desired criteria in a probe, it is easy to check them against a specific object. But if those criteria are different than those used to describe an object in memory, recall won't happen.

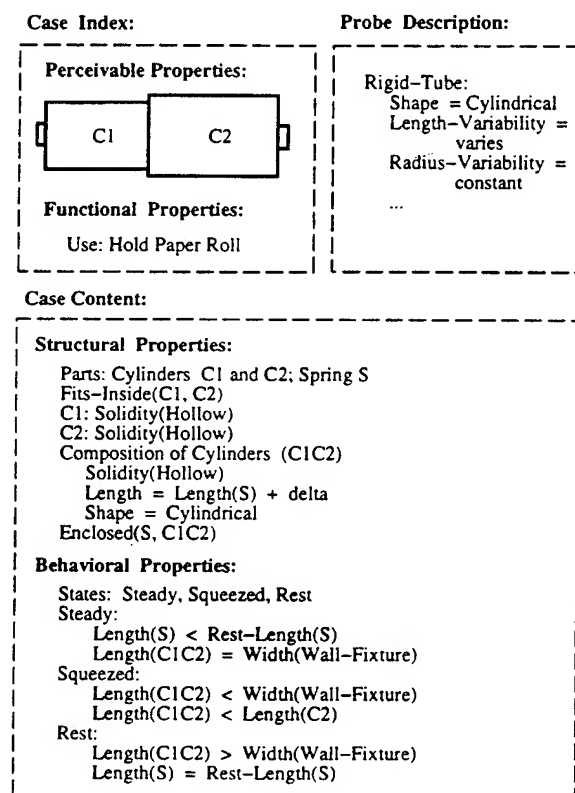


Figure 1: The many representations of a toilet paper holder

Figure 1 shows this mismatch. The collapsible tube, as

described after manipulating the springs (see the PROBE DESCRIPTION in Figure 1), has the structural property that its shape is *cylindrical* and the behavioral property that its length can *vary*. The toilet paper holder, on the other hand, is indexed in memory by a combination of its *Functional Properties* and *Perceivable Properties*, shown as INDEX in Figure 1. Thus, we cannot retrieve the CASE CONTENT, namely the *Structural Properties* and *Behavioral Properties* of the toilet paper holder by using the PROBE DESCRIPTION.

What facilitates recognition of the opportunity in the environment, i.e., recognition that the toilet paper holder can fulfill the role of collapsible tube? On the store's shelf, one can see the shape of the device. Recognition procedures perceive that it is a collapsible tube, which matches the description from the retrieval probe and presumably the label that designates what needs to be encountered to unblock the suspended goal.

But what processes direct recognition procedures to attend to the toilet paper holder on the store's shelf? And what mechanisms allow matching of something in the environment to a goal that is no longer active? We know that memory search is incremental and that when our memories can't retrieve what we are asking them for, we redescribe what we are looking for and try again. But when we aren't making headway, we postpone additional retrieval until more information is gathered and pursue other retrieval strategies or subgoals (Williams & Hollan, 1981, Norman & Bobrow, 1979, Kolodner, 1984). Similarly (and implied by predictive encoding), we suspend reasoning subgoals and subproblems that depend on postponed retrieval strategies and unmatched probes, marking them with criteria that, if encountered, predict that they should be reopened (Patalano, Seifert and Hammond 1992).

We propose that when an active subgoal (subproblem) is suspended, the subgoal and its criteria remain in working memory's working store for some limited time. We further propose that goals suspended in the working memory continuously monitor the environment, looking for matches in the environment to the specified criteria. Furthermore, we suggest that there are only a small number of these *active* goals. A computational model will provide more detail on these limitations.

A Memory Model

The Memory Architecture

The major component of our computational model (presented in Figure 2) is a working memory (WM), which communicates with both long-time memory (LTM) and perceptual processes and keeps track of recent reasoning context. As Barsalou (1992) suggests, the working memory mediates between short-term memory (STM) and the activated part of LTM. But we add significantly to Barsalou's conception. First, we give the WM a structure. Second, the structure integrates components of STM with activated portions of LTM and with perceptual mechanisms and stores. Third, this integrated component acts as a buffer for LTM. It is the place where LTM's components are manipulated and adapted. Fourth, we add a control unit (matcher), which can match (1) the current artifact being reasoning about or (2) all the suspended problems against the LTM representation of the current item presented to the Recognizer.

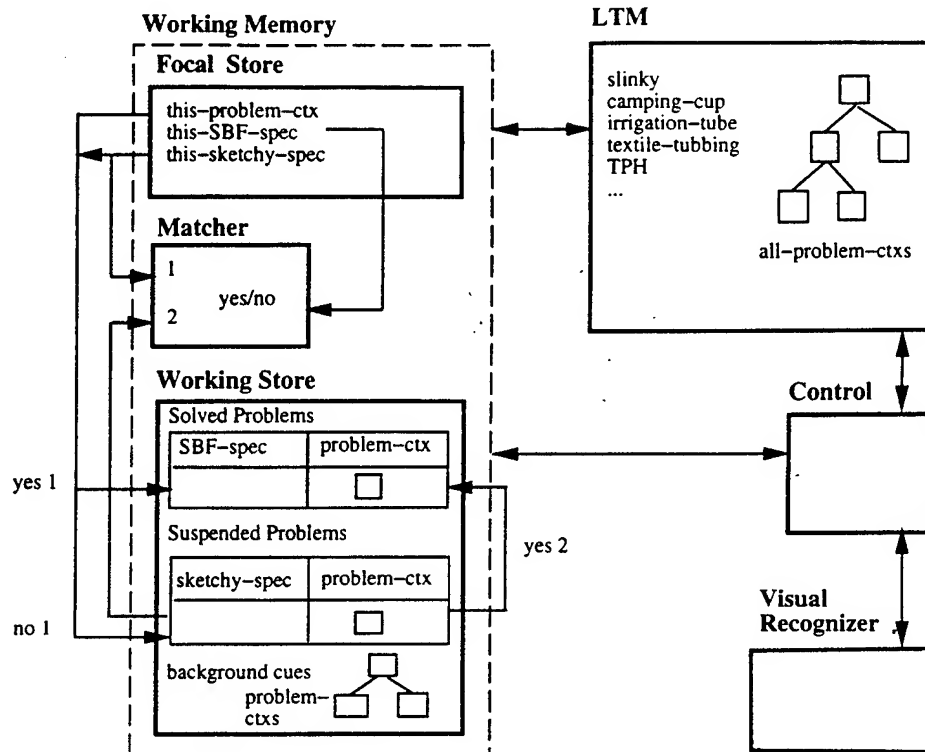


Figure 2: The Memory Architecture

The working memory that emerges has three parts: (1) Focal Store (FS), (2) WM Control unit (the only part of the control unit currently relevant is the Matcher), and (3) Working Store (WS).

The Focal Store (FS) holds three items: (1) the current goal of the reasoner (THIS-PROBLEM-CONTEXT), (2) the current object, artifact or idea being reasoned about (THIS-SKETCHY-SPEC, which is similar to the PROBE DESCRIPTION in Figure 1), and (3) the representation of the current item presented to the Recognizer, module (THIS-SBF-SPEC), which is retrieved from LTM according to the specification generated by the Recognizer.

The working store is more interesting and has four parts.

1. A connected graph of related unsolved subproblems, represented as subgoals and the contexts in which they are applicable, called *problem contexts* (represented as small rectangles in the figure). This graph might be a subset of a problem decomposition stored in LTM when the problem was previously considered, it may have been created during the reasoning session, or it may be a combination of the two. The goal of the reasoning session is to find a solution for the whole group of related problems.
2. Background cues, which provide a history of concepts, descriptions, features, and objects that have been considered during reasoning
3. A list of Suspended Problems, each represented by a problem context that includes the relevant subgoal, the context in which it is being considered, and the still-incomplete solution description (SKETCHY-SPECS). More specifically,

the representation of suspended subproblems is modeled after the content of problem contexts in design. A problem context in design, and a suspended subproblem in working memory, includes (1) a set of goals and partially ordered constraints that solutions should satisfy; (2) a set of options, or alternatives for achieving those goals²; and (3) a set of relationships describing how the options satisfy the constraints. These sets are incomplete and contain as much as has been considered so far in addressing the goals.

4. A list of Solved Problems, consisting of problem contexts for which solutions (SBF-SPECS) have been found. These problem contexts have the same structure as do suspended subproblems, but their solution descriptions are complete.

This working memory structure, in effect, keeps track of the part of LTM activated during a reasoning session. At most, then, the retention time of working memory is a few hours, requiring only a limited capacity (more work is needed before speculating on how big).

The working store accommodates several subproblems (PROBLEM-CTXs), which ideally are related, at the same time. These subproblems are approached one at a time, and if the current one cannot be solved, it is transferred to the list of Suspended Problems. Solved problems are transferred to the Solved Problems queue. A suspended problem is characterized by a non elaborated specification (SKETCHY-SPEC), which

²In our example the options set included a telescope, a camping cup and a slinky. This initial set of options was gathered by probing the LTM with a set of indexes relevant to the context goals.

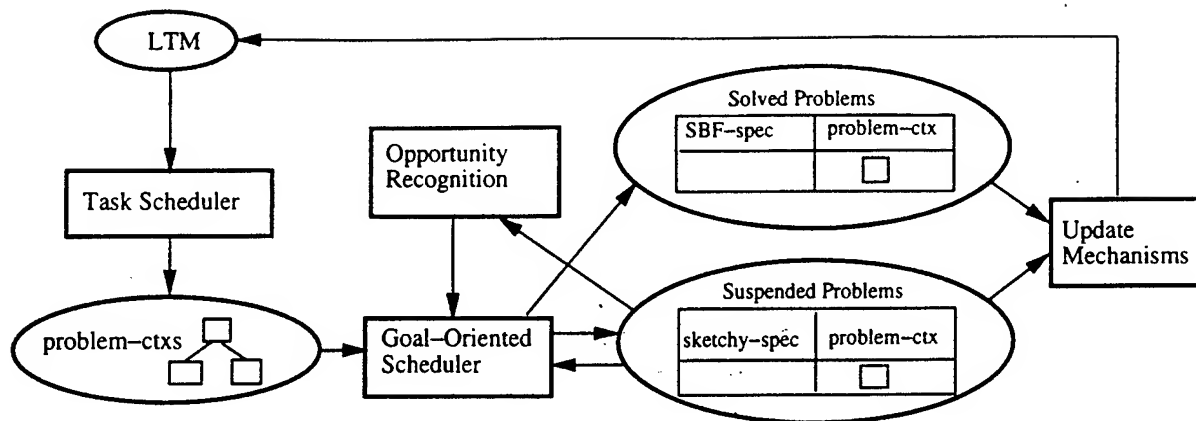


Figure 3: The Processing Algorithm

cannot be used as a successful index in the LTM. A problem is considered solved when the Matcher module recognizes that something in the environment or created on the fly fulfills the requirements formulated in a SKETCHY-SPEC.

The whole system is monitored by a global Control module, which is responsible for the flow of problem contexts between working memory, LTM, and the recognizer module, which perceives the world. When a reasoning session ends, the control module makes sure that relevant information from the WM updates the structures in LTM.

The Processing Algorithm

Mediation between working memory, long-term memory, and perceptual processes are key to working memory's functioning. Four control components (see Figure 3) are important to using working memory well: (1) The task scheduler loads a graph structure (a set of related subproblems) in the Working Store. (2) The goal-oriented scheduler uses the graph structure and the sets of suspended and pending problems to choose what to do next. Among other things, it suspends subproblems when no headway is being made; it reinstates them when their indexes (specs) are matched. (3) Opportunity recognition procedures notice opportunities to reinstate suspended goals and send messages to that effect to the scheduler. This is accomplished by having perceptual functions (the object recognizer is the only one of these in the scheme presented) focus their attention based on the sketchy specs recorded in suspended subproblems. For example, the object recognizer seeks to identify objects whose descriptions *partially match* the sketchy specs associated with suspended subproblems. When such an object is seen, the recognizer asks inference procedures if they can *quickly* determine if the object has other properties specified in the sketchy spec. If so, the opportunistic component notifies the goal scheduler that a suspended goal ought to be reinstated. (4) Update mechanisms update the structures in LTM based on recordings in WM.

When a new problem is approached, a hierarchical structure is proposed for it. Sometimes the structure is already recorded in memory; sometimes it is on paper; sometimes it must be constructed – we don't consider that issue right now. As a

next step, a small group of related subproblems is brought into focus and loaded into WM. It is essential that this group is kept small, because potentially all of its components may become active during reasoning and the computational demand should be limited. Exactly how this choice of subproblems is made must still be discovered; one option is to bring in only the most connected set of related subproblems and only up to some small threshold.

In our example, we assume that the full problem (design of a quick transportation device) has been considered previously and that there are a set of subproblems recorded in memory. In the session we focus on, two subproblems are brought to attention and loaded in WM: the launching device problem and the cushioning material problem. The graph structure in WM has the full problem at the top and these two subproblems hanging off of it as sibling subproblems.

One of the subproblems is chosen for focus, and it is loaded into the Focal Store as the current problem (THIS-PROBLEM-CTX). Here, the launching device problem is chosen first. Reasoning procedures work on this problem until it is solved, in which case it is put into the solved problems queue, or until no progress can be easily made, in which case it is added to the queue of suspended problems. When the need for a collapsible tube emerged in solving the launching device problem, no useful device was recalled from LTM, nor was one seem immediately on the shelves of the store. Thus, this subproblem is suspended. The description created of the collapsible tube (the probe in Figure 1) is used as the SKETCHY-SPEC for this suspended problem. When a subproblem is suspended, a new problem is chosen to work on. Here, the cushioning subproblem is selected, and reasoning procedures begin working on it.

At the same time, perceptual functions are scanning the environment, looking particularly for things that partially match sketchy specs of suspended problems³. In our case, the object recognizer notices the toilet paper holder on the shelf of the

³In fact, the sketchy specs are related with the visual images of the options considered so far (telescope, camping cup and slinky in our example). Priming processes recognize some of these options when the visual recognizer is scanning the environment (sometimes spuriously). Indirectly, the associated sketchy specs are remembered.

store. The TPH matches the collapsible tube specification because it is cylindrical and a rigid tube. This is enough of a partial match to the recorded sketchy spec that it asks inference procedures whether the TPH also has variable length. A simple scan of the full representation of a TPH (i.e., the one in LTM that includes behavioral and structural information) supplies a positive answer (we know that a TPH has to be compressed in order to be assembled to provide support).

When a subproblem becomes unblocked due to new information becoming available, the goal scheduler unblocks the suspended problem and asks reasoning mechanisms to proceed in reasoning about it. This is what happens with the launching device problem.

Status and Open Issues

The working memory model discussed here is implemented as part of the IMPROVISER system (Wills & Kolodner 1994a, 1994b). Our original intent was to extend IMPROVISER to allow it to handle and maintain multiple pending problem contexts. However, we soon realized that handling multiple problem contexts was a memory problem and that the mechanism that would allow that could also be used to explain at least some cases of opportunity recognition. We suspect that this approach will also provide us with ways of explaining forgetting during a long reasoning session and the "freshness" that reasoners feel when coming back to a problem after letting it rest for several hours or days. But more exploration is needed before we have good explanation for either of these phenomena. Indeed, we don't yet have a full explanation of the constraints on memory in handling multiple contexts and in maintaining control of the active goals involved in opportunistic recognition. We do believe, however, that we have proposed a framework within which these questions can be answered quite nicely. We look forward both to continued computational modeling and continued experimentation on people to answer these questions.

Acknowledgements

This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234. Special thanks to Linda Wills for helpful discussions about the IMPROVISER system. We thank Ashwin Ram, Hari Narayan, Linda Wills and our anonymous reviewers for their comments on this research.

References

- Barsalou, L. (1992). *Cognitive Psychology*. Lawrence Erlbaum Associates.
- Birnbaum, L. (1986). *Integrated Processing in Planning and Understanding*. PhD thesis, Yale University.
- Birnbaum, L. & Collins, G. (1984). Opportunistic Planning and Freudian Slips. *Proceedings of the Sixth Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates.
- Dehn, N. (1989). *Computer Story-Writing: The role of Reconstructive and Dynamic Memory*, PhD thesis, Yale University.
- Grimson, W.E.L. (1990). *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press.
- Hammond, K., Converse, T., Marks, M. & Seifert, C.M. (1993). Opportunism and Learning. *Machine Learning*, 10 (pp. 279-309).
- Hayes-Roth, B. & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science* 3 (pp. 275-310).
- Kolodner, J. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*. Lawrence Erlbaum Associates.
- Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann (pp. 369-382).
- Kolodner, J. & Wills, L. (1993a). Case-based Creative Design. *AAAI Spring Symposium on AI and Creativity*. Stanford, CA. March 1993.
- Kolodner, J. & Wills, L. (1993b). Paying Attention to the Right Thing: Issues in Case-Based Creative Design. *AAAI Case-Based Reasoning Workshop* (pp. 19-25).
- Mayer, N.R.F. (1970). *Problem Solving and Creativity: In individuals and Groups*. Brooks/Cole Publishing Company, (pp. 162-175).
- Norman, D.A. & Bobrow, D.G. (1979). Descriptions: An intermediate stage in memory retrieval. *Cognitive Psychology* 11 (pp. 107-123).
- Ram, A. & Hunter, L. (1992). The Use of Explicit Goals for Knowledge to Guide Inference and Learning. *Journal of Applied Intelligence*, 2(1) (pp.47-73).
- Patalano, A., Seifert, C. & Hammond, K. (1993). Predictive Encoding: Planning for Opportunities. *Proceedings of the Fifteenth Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates (pp. 800-805).
- Smith, S.M. & Blankenship, S.E. (1991). Incubation and the persistence of fixation in problem solving. *American Journal of Psychology*, Vol.104, No. 1 (pp. 61-87).
- Williams, M. & Hollan, J. (1981). The process of retrieval from very long time memory. *Cognitive Science* 5 (pp. 87-119).
- Wills, L., Kolodner, J. (1994a). Towards More Creative Case-based Design Systems. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. Seattle, WA.
- Wills, L., Kolodner, J. (1994b). Explaining Serendipitous Recognition in Design. *Proceedings of the Sixteenth Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates (pp. 940-945).

Cases, Reasoning and Bell's Telephone¹

Marin D. Simina and Janet L. Kolodner

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{marin, jlk}@cc.gatech.edu

This poster investigates memory issues that influence long-term creative problem solving and design activity, taking a case-based reasoning (Kolodner & Wills, 1993) perspective. Our exploration is based on a well-documented example: the invention of the telephone by Bell (1908). But to understand this act of creative design, we have to analyze Bell's earlier research goals. We abstract Bell's research method and the reasoning mechanisms he used that appear time and again in long-term creative problem solving. In particular, we identify an understanding mechanism used widely by those processes which rely on previous experience. Finally, we integrate the new mechanisms in a computer model, ALEC, which features creativity elements in case-based design.

Retrospectively, the obvious question related to the invention of the telephone is: what cognitive issues "delayed" the invention of the telephone till 1876? The basic principles of the telephone, electromagnetism and induction, had been known since 1831. Several inventors tried and failed to design the telephone, because they (1) relied too much on prevalent telegraphy practice, (2) ignored the basic principles of electromagnetism, and (3) gave up too soon. It looks like these inventors applied case-based reasoning poorly: they stuck to minor adaptations of telegraphy rather than reassessing the problem and analyzing it from a new perspective. In contrast, Bell reassessed the telephone problem as being acoustical and not electrical. When Bell was stuck in electrical details, he analyzed his telephony experiments using acoustical experiences and expertise.

Bell frequently interpreted and remembered his electrical experiments in terms of acoustics, that he could easily *perceive* without supplementary equipment. Consequently, in some cases he could *recognize opportunities* to solve suspended problems while pursuing other problems (i.e., the "undulatory current" was recognized while working on the multiple telegraph problem, by noticing peculiar acoustical effects). But not all the recognized opportunities fell in the above category. Sometimes, working on several problems *in the same period of time* facilitated knowledge transfer among them without any special perceptual elaboration (i.e., interleaved work on both the telephone and phonograph inspired the microphone design for the telephone).

Our exploration of creative design (Kolodner & Wills, 1993; Simina & Kolodner, 1995) provides an initial framework (i.e., the IMPROVISER system) for a more enriched and dynamic case-based reasoning able to explain some interest-

ing reasoning issues involved in the invention of the telephone. In this framework a designer evolves concurrently the design specification and a pool of alternatives under consideration, relying on his previous experience. But Bell's *understanding* and interpretation processes, exploring previous experience, seem far more complex than those handled by IMPROVISER. Bell used analogy when simple retrieval failed. When analogy also failed, Bell made new hypotheses by combining attributes of the (partial) design alternatives retrieved. This mechanism led to the famous hypothesis of the "undulatory current", by mixing properties of sound transmission and electrical currents. We realised that understanding is a memory issue that should play a more important role in a framework for creative design. IDEAL (Bhatta & al., 1994) stresses also the role of understanding in design. IDEAL revises its understanding of the problem based on generalizations built from specific experiences.

Based on Bell's case study, we are developing a computer model that integrates understanding and design problem solving to test our hypotheses about the role of case-based reasoning in long-term creative design. In our model the problem solving processes relying on previous experience call an understanding process, adapted from Moorman & Ram's (1994) creative understanding algorithm. The understanding process in turn may call design problem solving processes to achieve a better artifact understanding. We hope that a good understanding of creative design processes will help us build better tools to assist human designers.

References

- [Bell, Alexander Graham](1908). *The Bell Telephone: The Deposition of Alexander Graham Bell*, Boston: American Bell Telephone Co.
- Kolodner, J. & Wills, L. (1993). Case-based Creative Design. *AAAI Spring Symposium on AI and Creativity*. Stanford, CA. March 1993.
- Moorman, K. & Ram, A. (1994). A model of creative understanding. *Proc. Twelfth Annual AAAI Conference*. (pp. 74-79).
- Bhatta S., Goel, A. & Prabhakar, S. (1994) Innovation in Analogical Design: A Model-Based Approach. *Proc. Third International Conference on Artificial Intelligence in Design*, Lausanne, Switzerland. (pp. 57-74).
- Simina, M., Kolodner, J. (1995). Opportunistic Reasoning: A Design Perspective. *Proc. Seventeenth Conference of the Cognitive Science Society*. (pp. 78-83).

¹This research was funded in part by NSF Grant No. IRI-8921256 and in part by ONR Grant No. N00014-92-J-1234.

Cases, Reasoning and Bell's Telephone

Marin Simina

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
(404) 894-5612
marin@cc.gatech.edu

Abstract

This paper investigates memory issues that influence long-term creative problem solving and design activity, taking a case-based reasoning perspective. Our exploration is based on a well-documented example: the invention of the telephone by Bell. But to understand this act of creative design, we have to analyze Bell's earlier research goals. We abstract Bell's research method and the reasoning mechanisms he used that appear time and again in long-term creative problem solving and design. In particular, we identify an understanding mechanism used widely by those processes which rely on previous experience. Finally, we integrate the new mechanisms in a computer model, ALEC, that acts creatively in case-based design.

1 Introduction

This paper investigates memory mechanisms that influence long-term creative problem solving and design activity, from a case-based reasoning (CBR) perspective (Kolodner, 1993). Our exploration is based on a well documented example: the invention of the telephone (Bell, 1908). Retrospectively, the obvious question related to the invention of the telephone is: what cognitive issues "delayed" the invention of the telephone till 1876? The *basic principles* of the telephone, electromagnetism and induction, had been known since

1831. Several inventors tried and failed to design the telephone, because they (1) relied too much on prevalent telegraphy practice, (2) ignored the basic principles of electromagnetism, and (3) gave up too soon. It looks like these inventors applied CBR poorly: they stuck to minor adaptations of telegraphy rather than reassessing the problem and analyzing it from a new perspective. In contrast, Bell reassessed the telephone problem as being acoustical and not electrical. When Bell was stuck in electrical details, he analyzed his telephony experiments using acoustical experiences and expertise.

Bell's research plan, used to generate new learning goals, was much like that of other inventors. When Bell identified an unexpected function of a device, he tried to understand it in terms of his knowledge. When his understanding process failed, he generated new learning goals (Ram, 1991). But Bell brought a new perspective to the analysis of telephony by relying on his acoustical knowledge to generate new hypotheses. In analogy with acoustical sound transmission, Bell generated the electrical "undulatory current" hypothesis, essential for the design of the telephone. Bell frequently interpreted and remembered his electrical experiments in terms of acoustics, that he could easily *perceive* without supplementary equipment. Consequently, in some cases he could *recognize opportunities* to solve suspended problems while pursuing other problems (i.e., the "undulatory current" was recognized while working on the multiple telegraph problem, by noticing peculiar acoustical effects). But not all the recognized opportunities fell in the above category. Sometimes, working on several problems *in the same period of time* facilitated knowledge transfer among them without any special perceptual elaboration (i.e., interleaved work on both the telephone and phonautograph inspired the microphone design for the telephone).

Our exploration of creative design (Kolodner and Wills 1993a, Simina and Kolodner 1995) provides an initial framework (i.e., the IMPROVISER system) for a more enriched and dynamic CBR able to explain some of the reasoning issues involved in the invention of the telephone. In this framework a designer evolves concurrently the design specification and a pool of alternatives under consideration, relying on his previous experience (see Figure 1). Each new design alternative is evaluated mainly in relation to the *current goal* (i.e., the current design problem). But the evaluation process may serendipitously generate state information relevant to other *suspended goals* (i.e., some related problems which could not be pursued and were postponed), in addition to critiquing the design alternative and detecting ambiguities and

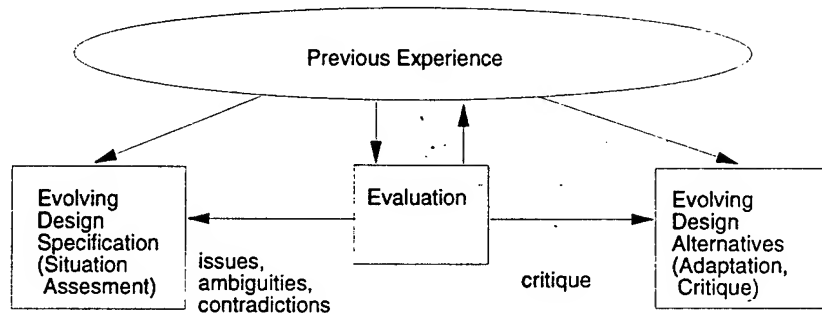


Figure 1: Framework for Case-based Creative Design

contradictions in the design specification. Our model therefore has a mechanism for detecting opportunities and for activating the suspended goals to which they pertain.

Our previous framework handled most of Bell's control strategy during telephony work. Bell used to work on several open problems during the same period of time. When he got stuck on one problem, he switched to another one. Sooner or later Bell revisited the same problem, but this time from a new perspective. Indeed, working on other problems in the mean time seemed to "refresh" his memory: interleaved work generated new learning goals and new background cues that were used to approach the same old problem in a new way. But those three main processes in our framework did not use effectively the available previous experience since they used simple retrieval to explain some important part of Bell's reasoning. In particular, Bell's understanding and interpretation processes seem far more complex than those handled by IMPROVISER. Bell used analogy when simple retrieval failed. When analogy also failed, Bell made new hypotheses by combining attributes of the (partial) design alternatives retrieved. This mechanism led to the famous hypothesis of the "undulatory current", by mixing properties of sound transmission and electrical currents. We realized that understanding is a memory issue that should play a more important role in our framework for creative design.

Based on Bell's case study, we are developing a model that integrates understanding and design problem solving, and building a computer program,

ALEC¹, to test our hypotheses about the role of CBR in long-term creative design. We characterize Bell's reasoning in terms of goals and plans, and we keep track of the active goals. The active goals can be triggered easily by small changes in the current state, while the reasoner is pursuing the current goal. Consequently, active goals can be satisfied opportunistically. Our computer model extends the memory architecture presented in Simina & Kolodner (1995).

2 Creative Design

2.1 Issues

A designer is charged with specifying the structure of an artifact that delivers some functions and satisfies some constraints (Chandrasekaran, 1990). Creative design adds two supplementary constraints: the designed artifact should be both *novel* and *useful*. But this simplistic characterization of the creative design task provides no clues in the processes involved. A novel artifact implies a novel design specification and it is difficult to separate the evolution of the design solution from the evolution of the design specification. The evolution of the design specification can be viewed as another design process (Tong, 1988) from an incomplete, contradictory and under-constrained specification to a better description, good enough for designing a novel artifact.

But how can designers get to an initial design specification? The exploration of the design space without a design specification (i.e., "tinkering"), can produce novel artifact ideas, but designers should be able to recognize and assess the usefulness of those ideas. However, recognizing opportunities is hard, especially when the domain is not well understood. For example, Elisha Gray, another inventor involved in the quest for the telephone, put on paper a specification containing all the fundamental principles of the telephone but failed to recognize their real significance. In a letter to Bell, Gray confessed:

I do not, however claim even the credit of inventing it, as I do not believe a mere description of an idea that has never reduced to

¹ Alec was Alexander Graham Bell's nickname

practice – in the strict sense of that phrase – should be dignified with the name invention. (Bruce, 1973, page 221)

The prerequisite of opportunistic behavior is the existence of suspended goals, goals that cannot be pursued in the current context and are postponed. As an example, Bell realized that he needed to produce an “undulatory current” in order to design the telephone, but he had no clue of how to produce such a current. Later, he was able to recognize such an “undulatory current” while working on the multiple telegraph problem.

The next question is: where these goals come from? Scientific curiosity is the main source of open problems and suspended goals. A designer may notice an anomalous situation, while pursuing a certain activity (e.g., understanding an artifact). If the anomalous situation is “interesting” (i.e., it reminds him/hers about other goals), the designer may encode in memory a new goal to understand the anomaly. Later, he/she may recognize an opportunity to better understand the anomaly. This is exactly what happened when Bell encountered Helmholtz’s Apparatus (relevant to his acoustic research). Bell was unable to understand some electrical details of the apparatus, but he returned to these details later when he recognized the opportunity to perform electrical experiments. By working on several related projects in the same period of time, creative designers incorporate “tinkering” opportunistically in long-term design: while working on one project, they might recognize design ideas for new projects. In Bell’s case, he generated the telephone idea while working on the multiple telegraph.

Once a designer has an initial design specification worth pursuing, he/she might generate and develop alternative design solutions. The most common methods for design use three subtasks: propose, critique, and modify (Chandrasekaran, 1990). The issue here is how to implement these subtasks to explain creative behavior. We are interested to find how designers use well-known designs in novel ways or how they engage in cross-domain transfer of abstract design ideas. In particular, we want to identify the mechanisms that combine useful pieces of different artifacts in a new artifact.

2.2 A Possible Approach

Our approach to understanding long-term creative design was to use our previous framework (Wills and Kolodner 1994, Simina and Kolodner 1995)

DTIC COULD NOT GET MISSING

PAGE FROM CONTRIBUTOR

The set of *Suspended Problems* is characterized by non-elaborated design specifications (sketchy-specs), which cannot be understood successfully by our system (ALEC). A problem is considered solved when the *Matcher* module recognizes that something in the environment or created on the fly fulfills the requirements formulated in a sketchy-spec.

The set of *Background Problems* contains problem contexts accessed recently. While devices in LTM are indexed normally by their function, alternative designs for problems in *Background Problems* can be accessed also by structural and behavioral properties. Bell's case study showed us that this is indeed the case. As an example, let's consider the episode when Bell had to design the microphone for the telephone:

The problem that then arose in my mind was, how to move a piece of steel in the way that the air was moved by the action of the voice. While this problem was in my mind, I was carrying on experiments with the phonautograph constructed from the human ear ... and it occurred to me that if such a thin and delicate membrane could move bones that were, relative to it, very massive indeed, why should not a larger and stouter membrane be able to move a piece of steel in the manner I desired? At once the conception of a membrane speaking telephone became complete in my mind; for I saw that a similar instrument to that used as a transmitter could also be employed as a receiver. (Bell, 1908 p. 39).

We notice here that the phonautograph design was recalled by a behavioral component (i.e. how to move heavy bones with a delicate membrane) which was not reflected in its function (i.e. record on paper the shape generated by different sounds). This was possible since the phonautograph design was present in the *Background Problems* set, due to its recent processing. This mechanism provides also an explanation of how pieces from an artifact may be selected for building a new artifact.

Each of the processes in Figure 2 (represented as gray rectangles) has the ability to inspect the Opportunity Agenda, to act upon relevant opportunities from the agenda, and to notice new opportunities for other processes, by recording them in agenda. The Opportunity Agenda contains a fixed set of generic opportunities which help to implement the standard CBR cycle. But once new opportunities are created by the External Recognizer (exogenous

opportunities) or by other internal process (endogenous opportunities), the flow of control may diverge from the standard CBR cycle. Since the control of our computational model is explicit (selection of opportunities from the agenda using heuristics), the model can keep a trace of the problem solving process and reflect on it, if necessary.

3.2 Processing Algorithm

Creative design usually evolves incrementally, during several design sessions. During each session, we address some old issues (questions) and we raise other new issues (see Figure 3).

TASK: incremental creative design

INPUT:

- an agenda of opportunity
- an incomplete, contradictory and underconstrained design specification (to build a novel and useful artifact) represented as a set of constraints ($K_{current}$)
- a set of design alternatives to build an artifact, satisfying the current set of constraints ($K_{current}$)

OUTPUT:

- an updated opportunity agenda
- an evolved set of constraints (K_{next}) representing the evolved design specification
- an updated set of design alternatives to build an artifact satisfying the (current) set of input constraints (K_{next})

METHOD:

opportunity = pick the most relevant opportunity
CASE(recipient of opportunity) OF
 eval: Evaluate an artifact (A) with respect to a goal (G)
 alt: Elaborate alternative solutions
 spec: Elaborate design specification

Evaluate, Elaborate Solutions and Elaborate specification
use the Creative Understanding subtask.

Figure 3: Incremental Creative Design

Our view of creative design includes two concurrent processes: one for *evolving alternative solutions* and another one for *evolving the design spec-*

ification. These two processes evolve under the control of a third process, *evaluation*. Evaluation questions the advantages and disadvantages of each alternative and also the contradictions and ambiguities in the specification. The resolution of these questions serves to refine both the design specification and the pool of partial design alternatives. The output of a creative design session becomes the input of the next session and the structure of input/output becomes an invariant of the process (see Figure 3). We call this invariant structure a *problem context*².

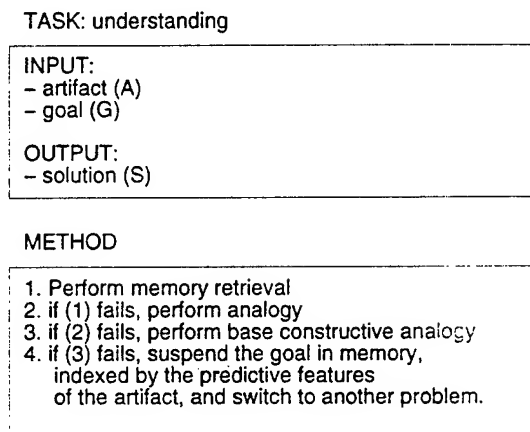


Figure 4: The Understanding Algorithm

In our view, the three main processes involved in creative design rely heavily on the use of previous experience. Each of these processes has to understand the current situation in order to act creatively. But if these processes perform only simple understanding based on classical memory retrieval, our case-based reasoner will fail to generate novel artifacts. Other more complex reasoning is needed.

As an example, when Bell tried to understand an electrical device (i.e., Helmholtz's Apparatus) without having enough electrical experience, simple retrieval failed to retrieve useful electrical information from his memory. Next, he relaxed the constraints and looked for devices which performed

²A problem context extends the question, options, criteria (QOC) notation presented in MacLean et al. (1991).

the same acoustical functions (analogy). This time he could use his whole acoustical expertise in understanding the electrical device. But sometimes this was not enough since the analogical mapping failed. When Bell made an analogy between speech transmission through air and sound transmission through electrical wires, standard analogy failed since electrical currents could transmit only the pitch of the voice, but not its amplitude. Sometimes it is possible to build dynamically a base within a given domain if the reasoner possesses some expertise in both target and intended base domain, by combining desirable attributes from several bases and evaluating the new base using knowledge from the target domain. Moorman and Ram (1994) call such a technique *base-constructive analogy*. Base-constructive analogy models the reasoning used by Bell in advancing the "undulatory current" hypothesis, by mixing properties of voice transmission through air and sound transmission through electrical wires. When base constructive analogy failed, Bell simply encoded predictively the understanding problem in memory and started working on another problem.

The new understanding algorithm, which supersedes our simple retrieval algorithm, is presented in Figure 4. It borrows from Moorman and Ram (1994), with some differences. The algorithm retrieves first all the devices that have *all* the attributes of the artifact to be understood. If retrieval is successful, then the new artifact may be understood in terms of one of the retrieved artifacts. Otherwise, the algorithm tries analogy, by relaxing the retrieval constraints (only the predictive attributes of the artifact's function are used in retrieval). If something is retrieved, then the algorithm performs understanding using structural mapping (Gentner, 1983). Otherwise, the algorithm tries base-constructive analogy. If base-constructive analogy fails, our understanding algorithm does not reformulate the problem (as in Moorman and Ram's algorithm), but suspends the goal in memory (indexed by the predictive features of the artifact) and gives control to the goal scheduler to switch to another problem. Problem reformulation is a separate process in our framework, which is driven by an evaluation process. The evaluation process relies on real world interaction (i.e., experimentation), in order to decide how to reformulate the problem. A simple "mental" simulator, using structural mapping, cannot model the complex reasoning that lead to the invention of the telephone.

4 Discussion

The proposed computational model makes explicit the role of understanding during design problem solving. Moorman and Ram (1994) have addressed the complementary problem of making explicit the role of design during story understanding. In contrast with Moorman and Ram's model, our model stresses the importance of real-world evaluation (i.e., experimentation) for design problem solving. Clement (1989) emphasizes also the role of experimentation for problem solving via model construction, but he does not propose a mechanism for constructing a new artifact by merging pieces from other candidate artifacts. But our model does not provide introspective reasoning on the reasoning trace (Ram and Cox, 1994), which also might be responsible for creative behavior.

5 Conclusions

We have presented a computational model for creative design, based on the utilization of an understanding algorithm for the ALEC's processes which rely on previous experience. The new model was able to offer a plausible explanation for some relevant episodes involved in the design of the telephone. In order to perform analogical reasoning, both the function of the base artifact and other behavioral attributes can be used to retrieve a potential target. The function of the artifact is used to retrieve artifacts from long-term memory, while the behavioral attributes may be used to retrieve artifacts in the working memory.

6 References

- [Bell, Alexander Graham](1908). *The Bell Telephone: The Deposition of Alexander Graham Bell*, Boston: American Bell Telephone Company.
- Bruce, R.(1973). *Bell: Alexander Graham Bell and the Conquest of Solitude*, Little Brown & Co.
- Clement, J. (1989). Learning via Model Construction and Criticism. In Glover, G., Ronning, R, & Reynolds, C., *Handbook of Creativity: Assessment, Theory and Research*, (pp. 341-381)

- Goel A. (1992). Integrating Case-Based and Model-Based Reasoning: A Computational Model of Design Problem Solving. *AI Magazine*, 13(2) (pp. 50-54), Summer 1992.
- Kolodner, J. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*. Lawrence Erlbaum Associates.
- Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann (pp. 369-382).
- Kolodner, J. & Wills, L. (1993a). Case-based Creative Design. *AAAI Spring Symposium on AI and Creativity*. Stanford, CA. March 1993.
- Kolodner, J. & Wills, L. (1993b). Paying Attention to the Right Thing: Issues in Case-Based Creative Design. *AAAI Case-Based Reasoning Workshop* (pp. 19-25).
- Mayer, N.R.F. (1970). *Problem Solving and Creativity: In individuals and Groups*. Brooks/Cole Publishing Company, (pp. 162-175).
- Moorman, K. & Ram, A. (1994). A model of creative understanding. In *Proceedings of the Twelfth Annual AAAI Conference*.
- Norman, D.A. & Bobrow, D.G. (1979). Descriptions: An intermediate stage in memory retrieval. *Cognitive Psychology* 11 (pp. 107-123).
- Patalano, A., Seifert, C. & Hammond, K. (1993). Predictive Encoding: Planning for Opportunities. *Proceedings of the Fifteenth Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates (pp. 800-805).
- Ram, A. (1991). A theory of Questions and Question Asking. *The Journal of Learning Sciences*, 1(3 & 4) (pp.273-318)
- Ram, A. & Cox, M. (1994). Introspective reasoning using meta-explanations for multistrategy learning. In Michalski, R.S. and Tecuci, G. *Machine learning: A Multistrategy Approach, Vol. IV*
- Ram, A. & Hunter, L. (1992). The Use of Explicit Goals for Knowledge to Guide Inference and Learning. *Journal of Applied Intelligence*, 2(1) (pp.47-73).

- Simina, M., Kolodner, J. (1995). Opportunistic Reasoning: A Design Perspective. *Proceedings of the Seventeenth Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates (pp. 78-83).
- Wills, L., Kolodner, J.(1994a). Towards More Creative Case-based Design Systems. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. Seattle, WA.
- Wills, L., Kolodner, J.(1994b). Explaining Serendipitous Recognition in Design. *Proceedings of the Sixteenth Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates (pp. 940-945).

7 Appendix: A Case Study

This case study analyzes some important reasoning steps involved in the telephone invention. Bell's education was important for his future scientific goals. The study of Helmholtz's apparatus introduced Bell to electricity and was also a prerequisite for the invention of the multiple telegraph. Bell's work on the multiple telegraph inspired the invention of the telephone. The design of the telephone was officially completed in March 7, 1876, when the U.S. Patent No. 174,465 was issued.

7.1 Background

Bell was specially educated by his father to be a *teacher of articulation* specialized in the "visible speech" method (Bell 1908, page 7). The idea was to provide alternative forms of acoustic feedback for deaf mute students learning to speak. Consequently, Bell set himself the following agenda of thematic goals for his teacher role:

1. teach articulation (based on the "visible speech" method)
plan: provide visual feedback
2. learn "everything" about acoustics and speech
to improve the "visible speech" method
3. publish (communicate) original research

In 1864, Bell accepted a teaching position of articulation and music. As a teacher, Bell set himself the goal to understand the role of the articulatory system in the production of vowels. First, Bell noticed the existence of two cavities in the articulatory system. The two cavities had the acoustical *function* of resonance chambers, each characterized by its own pitch. After an elaborate set of experiments, Bell identified that "the vowel quality was produced by the resonance tones of the mouth-cavities mingling faintly with the tone of the voice".

This is where the interesting part of Bell's story began. Bell communicated this original result (one of his goals as a teacher) to the relevant research community. Surprisingly, Bell found:

*... that the experiments had already been made by Helmholtz,
and that Helmholtz had demonstrated the compound nature of the*

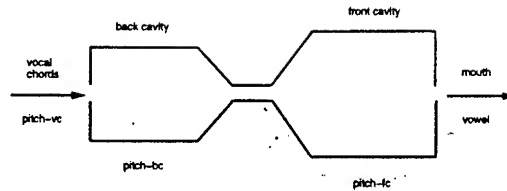


Figure 5: Bell's Articulation Model

vowel sounds by producing them artificially by a synthetical process. (Bell 1908, page 8)

Bell considered the above information “interesting” for his goal learn “everything” about acoustics and speech (more exactly, Bell could use the apparatus built by Helmholtz to test his own speech theory). In order to satisfy this goal, Bell set himself the subgoal to understand Helmholtz's Apparatus. If we want to understand how the subgoal was generated, we can ask ourselves: how an agent (Bell in particular) can learn “everything” about a *specific* area? As a general method (heuristic), the agent should *detect* potential challenging problems and next he should try to *understand* them. If the agent detects interesting anomalies during the understanding process (surprises), he can use them to generate new learning goals. We will refer to the above method as *Bell's research method*.

7.2 Understanding Helmholtz's Apparatus

7.2.1 Mapping

Both Helmholtz's Apparatus and Bell's Speech Model performed the same acoustic function (vowel production). Consequently, Bell tried to understand Helmholtz's Apparatus by comparing and contrasting it to his speech model.

The structure of both devices had little physical resemblance, but the function of the structural elements mapped straightforwardly:

For example, he [Helmholtz] would cause the simultaneous vibration of three tuning-forks of different pitches - one of these would represent the pitch of the voice - and this fork he caused

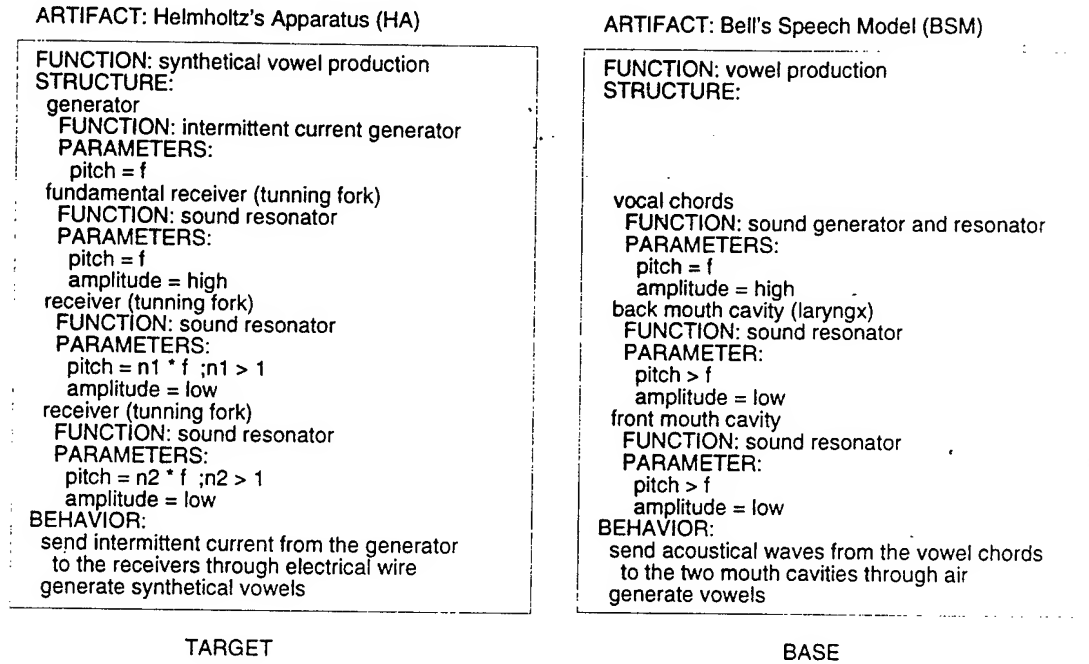


Figure 6: Mapping BSM and HA

to vibrate in front of a resonator tuned to its own pitch, so as to cause it to produce a loud musical tone. The other two forks corresponded in pitch to the front and back cavities of the mouth in uttering some vowel sound. The effect upon the ear was as though someone sang a vowel sound. (Bell 1908, page 8)

In Figure 6 we present the analogy between Bell's speech model and Helmholtz's Apparatus. The figure represents only the relevant elements for the analogical mapping. Initially, the target was identified as having the same function as the source. As a next step, the acoustical function of the structural components was mapped. Finally, individual parameters (attributes) have been mapped.

7.2.2 Surprises

But Bell had also surprises in understanding the Apparatus. According to *Bell's research method* (see above), these surprises were used to generate further learning goals. First, Bell noticed that Helmholtz used a variable number of harmonic receivers (not necessarily two). This surprise generated a learning goal: find why more harmonics are necessary in order to generate synthetical vowels. This learning goal was indexed by the structure of HA. The explanation provided by Helmholtz was that accurate (synthetical) reproduction of some vowel sounds needs a big number of harmonics (according to the Fourier analysis). The explanation was associated with the structure of HA.

Next, by studying the behavior of HA, Bell had another surprise, namely that electrical wires can be used instead of air for transmitting sounds at distance. The creative understanding algorithm can provide some clues in Bell's reasoning. Bell *hypothesized* that electrical wires (in the context of HA) and air have the same function (both transmit sounds), but was unable to fulfill the mapping successfully. Initially, Bell found that he does not have the electrical knowledge to fulfill the mapping, so he suspended the problem:

Helmholtz kept his forks in vibration by means of electro-magnets and a voltaic battery; but I found that I had not sufficient electrical knowledge to understand the arrangement used by Helmholtz. I therefore determined to study electricity. (Bell 1908, page 8)

7.2.3 "Odds and ends"

At the end of the compare and contrast between HA and Bell's Model, the generator used in HA did not have a counterpart in BM (so the mapping was not perfect). In order fully understand HA, Bell had to remember another relevant experience. First Bell had to assess the situation, namely to find quickly what (acoustical) peculiarities characterized the generator. Bell could notice the resonance phenomenon (same frequency at the generator and at the fundamental receiver). Using this cue, Bell was reminded of "the piano experience":

... I was familiar with the fact that the strings of a piano could be set into vibration by the action of the voice. For example, depress the pedal of a piano so as to rise the dampers from the

strings, and then sing into the instrument. That string which corresponds in pitch to the note will respond. (Bell 1908, page 36)

The immediate goal was to use this experience to understand the role of the generator. However, since this experience was remembered in the service of the goal understand HA, Bell tried to map the past experience with HA as much as possible (opportunistic reasoning). The analogy between "the piano experience" and Helmholtz's Apparatus is presented in Figure 7.

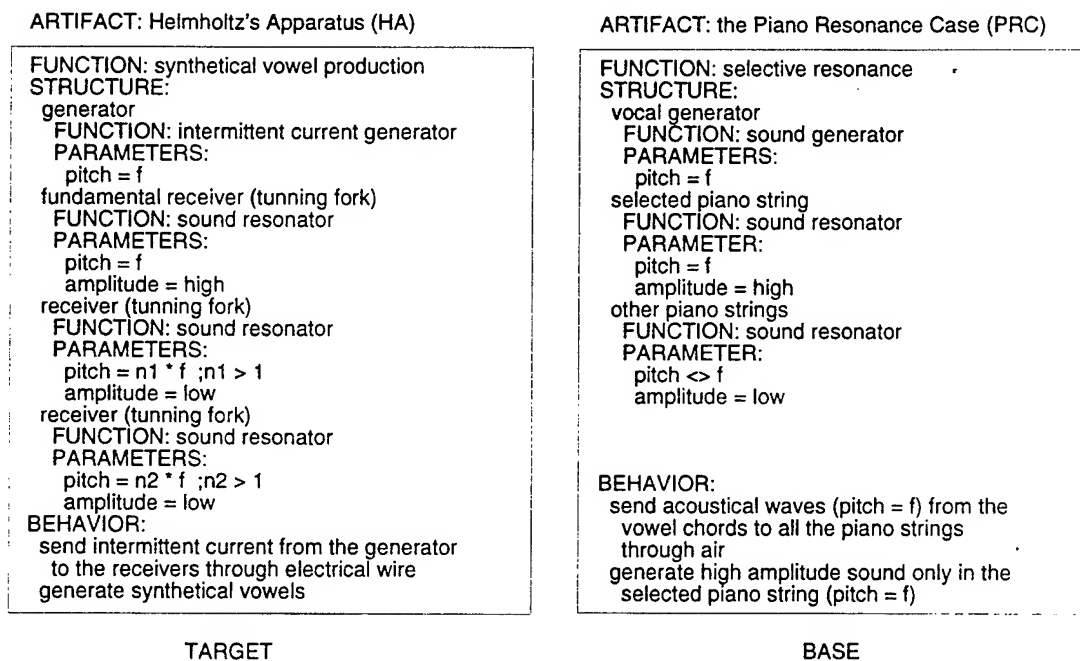


Figure 7: Mapping PRC and HA

This time the generator from HA mapped with the vocal generator and the fundamental receiver from HA mapped with the selected piano string. Moreover, the nonresonating harmonic receivers from HA mapped with some non-resonating piano strings. The medium of transmission was electrical wire for HA and air for PEM, but this was not a surprise anymore. Since the map-

ping was successful, all the explanations about the role of the vocal generator from PEM could be carried over to the role of the *generator* in HA.

In conclusion, Bell noticed that HA performs *also* the function of selective resonance (or demultiplexer), depending on the frequency of its intermittent current generator. Bell learned a new index for HA, namely selective resonance.

7.3 Aha! Multiple telegraph

In 1870's, a challenging problem in the telegraphic industry was the invention of a "multiple telegraph", whereby several messages could be transmitted simultaneously over the same wire. In 1872 the telegraphic industry adopted the Stearns' duplex (two-message) system, and it was looking forward the invention of a four- or eight-message system. Fame and fortune awaited the inventor of a multiple telegraph.

After reading a newspaper article³ about the quest for a multiple telegraph, Bell realized that he could design a multiple telegraph based on his knowledge of Helmholtz's Apparatus. We present a possible analysis of this episode.

While reading the newspaper article about the multiple telegraph, Bell tried to *understand* how the system worked. Bell assessed the problem as follows: (1) it involves electricity and telegraphic equipment and (2) a message is coded and transmitted over electrical wire to be received *selectively* only by one receiver.

The first requirement (1) brought into WM Helmholtz's Apparatus (since this was the only electrical device in which he was interested in the past). Consequently this created a context to understand (2). While addressing the problem of transmitting messages through electrical wires, Bell *recalled* the similarity of sounds and electrical currents (in the context of HA). Since nothing else was recalled, the only chance to understand further the problem was to reformulate it in terms of acoustics, where Bell was expert.

Bell reformulated the problem as involving *selective resonance*: a sound is coded and transmitted through air to be received selectively by only one recipient. The "piano experience" was remembered and it offered a solution to the problem. Namely, if a person sings *intermittently* close to the

³In the newspaper where Bell advertised his speech lessons for deaf students.

piano strings at a particular pitch, only the piano string with corresponding pitch will respond *intermittently*. What about several people? Bell mentally simulated that no interference will appear. Consequently, Bell assessed that the problem had a solution in the acoustical domain. By putting the electrical constraint in the context of the "piano experience", Bell retrieved Helmholtz's Apparatus. Since HA provided only a partial solution to the multiple telegraph problem, Bell had to guide the design of the multiple telegraph by using the acoustical model developed in the context of the "piano experience". In order to send one message to a specific recipient, Bell had to introduce a telegraphic key in the HA generator's circuit. For sending several messages, several generators of different pitches had to be introduced in the circuit. Bell simulated mentally his design and concluded its validity.

Bell developed the acoustic solution to MT in the context of understanding the newspaper article about the multiple telegraph. However, this solution enabled a state that activated other goals. The result was "interesting" for Bell's thematic goal learning "everything" about acoustics and also for the more human goal to become rich and famous, mentioned in the newspaper article. Since there was no goal conflict, Bell decided to pursue further the goal design a multiple telegraph. After designing the theoretical model of the multiple telegraph, Bell generated the goal implement the multiple telegraph, but he needed to learn more electricity first.

One plan to satisfy the goal learn more electricity is read electricity manuals. Bell found an electricity book by Baile and started to learn electricity to fulfill his goal to implement the multiple telegraph. In this book, he could read about:

*a series of vibrating plates, answering to the strings of a harp,
... each of which vibrates when struck by a particular sound,
and sends off electricity to create at the end of the line the same
vibrations in a corresponding plate. (Bruce, 1973, page 122)*

Bell could recognize that the *structure* and the *behavior* of the above design was relevant for his *active* goal implement the multiple telegraph. Consequently, he added the Baile's design as a *design alternative* to the goal implement the multiple telegraph. Bell noticed also that the new design transforms sounds into vibrations via intermittent currents. This function was relevant to Bell's goal of teach articulation by providing visual feedback (since the mechanical vibrations could be perceived visually). Consequently, Baile's

design became also a design alternative for the goal to build technology to provide visual feedback . But Bell simulated mentally that: (1) the vibrations of the transmitter's plates produced by human voice are too small to be perceived, and (2) the intermittent currents used cannot transmit the amplitude of the voice to the receiver's plates. These observations were attached to the Baile's design and to the goal *visualize speech*.

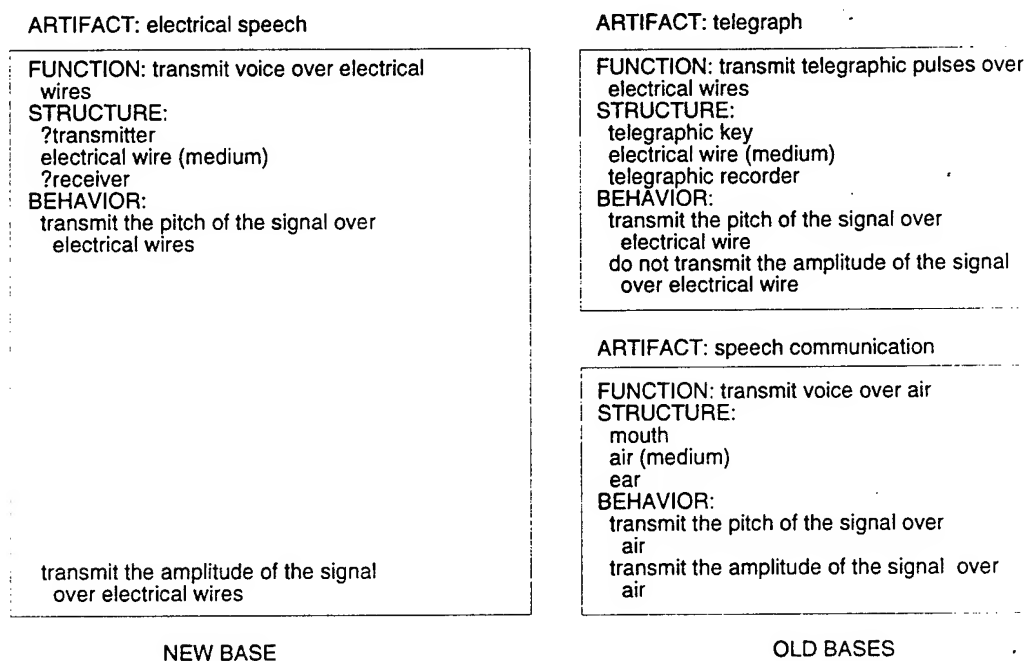


Figure 8: Base constructive analogy

In the summer of 1874 Bell started experiments with the *phonautograph*, a device built from a dead man's ear for visualizing speech⁴. Of course, this goal was generated in the service of the plan provide visual feedback for deaf people. During these experiments, Bell was struck by the way sound waves acting on a tiny membrane could move relatively heavy bones. This information contradicted his previous mental observation made for Baile's design.

⁴This involved fastening a delicate stick to a moving ear bone and recording traces of complex waves on smoked glass, while someone spoke into the ear membrane.

Since Baile's design was an alternative of the current active goal, Bell had an opportunity to revisit the design. Only one unsatisfied constraint remained, namely that the intermittent currents used in Baile's design could not transmit the amplitude of the voice. Based on the analogy between electrical currents and sound transmission, Bell *supposed* the existence of an "undulatory current", by mixing relevant properties of electrical currents and sound transmission (see Figure 8), and simulated Baile's design again. This time, a new function of the device emerged⁵: transmission of voice at a distance (Bruce, 1973 p.122). The new function was considered acoustically "interesting" and Baile's design was mentally adapted into the "harp apparatus". The harp apparatus was indexed under the "undulatory current" hypothesis. The hypothetical design of the telephone was predictively encoded in LTM and Bell switched his focus of attention to the Multiple Telegraph design.

7.4 Aha! Telephone

In June 2, 1875, Bell was performing experiments with the Multiple Telegraph. Due to an implementation mistake, Bell noticed that one of the reeds produced a sound which reproduced both the pitch and the amplitude of the generator, that is the device transmitted an "undulatory current" (produced by induction). The telephone design (harp apparatus) was retrieved immediately. Moreover, the incident proved that only one reed could transmit all the harmonics of a signal and that the induction currents generated are significant:

These experiments at once removed the doubt that had been in my mind since the summer of 1874, that magneto-electric currents generated by the vibration of an armature in front of an electro-magnet would be too feeble to produce audible effects that could be practically utilized for the purposes of multiple telegraphy and of speech-transmission. (Bell, 1908 p.59)

The above experiment switched Bell's focus of attention to the telephone design. Bell had already the main structural components to implement the telephone behavior. The next step was the design of the microphone:

⁵Baile's design was also an alternative for the Multiple telegraph, where the main function was transmission of information at distance.

The problem that then arose in my mind was, how to move a piece of steel in the way that the air was moved by the action of the voice. While this problem was in my mind, I was carrying on experiments with the phonautograph constructed from the human ear ... and it occurred to me that if such a thin and delicate membrane could move bones that were, relative to it, very massive indeed, why should not a larger and stouter membrane be able to move a piece of steel in the manner I desired? At once the conception of a membrane speaking telephone became complete in my mind; for I saw that a similar instrument to that used as a transmitter could also be employed as a receiver. (Bell, 1908 p. 39).

We notice here that the phonautograph design was recalled by a behavioral component which was not reflected in its function (i.e. record on paper the shape generated by different sounds). This was possible since the phonautograph design was present in the activated part of the long-term memory. Standard analogy was good enough to generate the design of the microphone, based on the the human ear.

GIT-CC-96/14

**MODEL-BASED ANALOGY
IN INNOVATIVE DEVICE DESIGN**

A THESIS
Presented to
The Academic Faculty

By

Sambasiva R. Bhatta

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in Computer Science

Georgia Institute of Technology
December, 1995

Copyright © 1995 by Sambasiva R. Bhatta
All Rights Reserved

GIT-CC-96/14

**MODEL-BASED ANALOGY
IN INNOVATIVE DEVICE DESIGN**

A THESIS
Presented to
The Academic Faculty

By

Sambasiva R. Bhatta

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in Computer Science

Georgia Institute of Technology
December, 1995

Copyright © 1995 by Sambasiva R. Bhatta
All Rights Reserved

DEDICATION

— To my wife Radha —

and

— To the memory of my brother Srinath —

ACKNOWLEDGMENTS

I am very grateful to my advisor, Ashok Goel. This thesis would not have been possible without his continuous intellectual encouragement, inspiration and guidance through out this work. I greatly appreciate his dedication to the betterment of my research and willingness to discuss issues. He has helped me in every step of this research with his insight, his questioning, his ideas, and his vision. At every point of crisis in my graduate career, he has not just one answer but always several that helped me make informed decisions. I appreciate his confidence in me even at times when I doubted myself. I thank him for his suggestive comments on several drafts of this document and the papers we have written together. He has been very patient in going through my writing again and again, and helped me improve my presentation. I sincerely thank his academic, financial, emotional, and personal support during my graduate career.

I am also indebted to my other committee members, Professors Richard Catrambone, T. Govindaraj, Janet Kolodner, Nancy Nersessian, and Ashwin Ram who have provided constructive criticism and insightful suggestions during this work. Each of them has brought a different perspective to my work which helped it become broader. Their questions from different angles have helped me think through issues clearly and clarify my ideas. Their suggestions of related work have helped me place my research better in the field of AI and Cognitive Science. They have always found time to meet with me and to read my thesis even in short notice. I really appreciate their understanding, support, and encouragement during this work. I like to specially thank Janet Kolodner for her thorough reading of multiple versions of my thesis and her comments on writing style and presentation. Her suggestions have helped me to improve my thesis tremendously.

I have greatly benefited from my discussions with several other faculty members and research staff of the AI and Cognitive Science group at Georgia Tech in different avenues. I would like to thank Linda Wills, Hari Narayanan, Mimi Recker, Tony Simon, Dorrit Billman, Larry Barsalou, Susan Bovair, and Kurt Eiselt. I would also like to thank Richard Billington for his help with my machine and software problems.

This research has also greatly benefited from many discussions with other members of the KRITIK research group. In particular, I would like to thank S. Prabhakar, Andres Gomez, Todd Griffith, Mike Pearce, Nathalie Grue, Paul Rowland, Khaled Ali, and Bill Murdock. Although Prabhakar has visited our group only for six months, his advice, encouragement, and moral support during that short time have helped me far beyond that period.

My special gratitude is to Kavi Mahesh, Eleni Stroulia, Murali Shankar, Justin Peterson and Mike Cox with whom I have shared the most of my graduate life. Without their company, it

would not have been fun and enjoyable. I specially thank Mahesh for his friendly and useful advice on many aspects, both academic and personal. I also thank Eleni for being such a "nice" office-mate. She has helped me to learn to be practical. Justin has been there to lighten up things with his "easy-go" attitude. As a colleague even now at NYNEX, I get to enjoy his humor.

Jeannie Terrell and Allyana Ziolk, both as secretaries of the AI group, have helped me with administrative things. I appreciate their help.

I would like to thank all my friends in and out of school that have given me constant moral support. I would like to specially mention my roommate Srinivas Doddapaneni for being a caring friend. I would also like to mention Venkatraman and Adhithi for their invaluable company. My long-time friend Sastry has been there with his encouragement in the final stages of my dissertation.

During the course of this work, I was supported by research grants from the Office of Naval Research (contract N00014-92-J-1234) and NSF, a CER grant from NSF (grant CCR-86-19886), and equipment donated by IBM, NCR, and Symbolics.

Without the constant encouragement from my technical director, John Martin, since I joined NYNEX Science & Technology, it would not have been possible to complete my dissertation. I greatly appreciate his understanding and flexibility with my schedule.

Last but not least, I like to thank my wife and best friend, Radha for her invaluable help in successfully completing the thesis. Without her encouragement, moral support, and cooperation, at the times when I most needed, this thesis would not have been possible. I really appreciate her immeasurable support in all times of my frustration. She has been very patient with me in waiting for the big moment. My family, especially my parents, have supported my venture into this journey since its beginning. They have also been very encouraging and patient as my journey became longer and longer. My brothers have also encouraged me in every possible way. I specially thank Ramana. I dedicate this thesis to Radha and to the loving memory of my brother Srinath.

CONTENTS

DEDICATION	i
ACKNOWLEDGMENTS	ii
LIST OF TABLES	viii
LIST OF FIGURES	ix
SUMMARY	xii

Chapters

I. INTRODUCTION AND OVERVIEW	1
1.1 Issues in Innovative Design	2
1.1.1 Research Problem	2
1.1.2 Research Hypothesis	3
1.2 Conceptual Framework	3
1.2.1 The Task: Design	4
1.2.1.1 Conceptual Design	4
1.2.1.2 Adaptive Design	4
1.2.1.3 Innovative Design	5
1.2.2 The Method: Analogical Reasoning	7
1.2.3 The Knowledge: Device Models and Design Patterns	9
1.2.4 Issues and Research Hypotheses	9
1.3 Computational Theory	10
1.3.1 The Task: Device Design	10
1.3.2 The Method: Model-Based Analogy	11
1.3.3 The Knowledge: SBF Models of Devices and Design Patterns	11
1.3.3.1 Design Patterns: GPPs and GTMs	13
1.3.3.2 Device Models and SBF Representations	15
1.3.4 Refined Issues and Hypotheses	15
1.4 The Computer Program: IDEAL	19
1.4.1 An Illustrative Example	20
1.5 Methodology and Evaluation	27
1.6 Organization of the Thesis	30
II. A COMPUTATIONAL THEORY OF MODEL-BASED ANALOGY	31
2.1 The Conceptual Design Task	33
2.2 Design Analogues, Device Models, and Design Patterns	33
2.3 The Subtasks of Model-Based Analogy	35
2.3.1 Retrieval of A Source Analogue	36
2.3.2 Transfer and Modification	37
2.3.2.1 Spawning of Adaptation Goals	37
2.3.2.2 Multistrategy Adaptation	38
2.3.3 Evaluation of the Solution	39
2.3.3.1 Internal Evaluation and Design Failures	39
2.3.3.2 External Evaluation and Design Failures	40

2.3.4	Learning from the Source and Target Analogues	41
2.3.4.1	Problem-Solving Failures and Knowledge Acquisition	41
2.3.4.2	Model-Based Learning of Design Patterns	42
2.3.5	Storage of the Target Analogue and the Design Pattern	44
2.4	Model-Based Analogical Design: An Illustrative Example	44
III.	THE CONTENT OF DEVICE MODELS AND DESIGN PATTERNS	57
3.1	Structure-Behavior-Function Models of Devices	58
3.1.1	Structure	58
3.1.2	Function	61
3.1.2.1	Behavioral State	63
3.1.3	Behavior	65
3.1.3.1	Behavioral State Transition	66
3.2	Primitive Functions	69
3.3	Substance Knowledge	70
3.4	Component Knowledge	72
3.5	Structure-Behavior-Function Models of Design Patterns	74
3.5.1	Generic Physical Processes	74
3.5.2	Generic Teleological Mechanisms	78
IV.	ANALOGUE MEMORY: MODEL-BASED INDEXING, ORGANIZA-	
	TION, RETRIEVAL AND STORAGE	80
4.1	Indexing and Organization of Design Analogue Memory	81
4.1.1	Functional Indexing and Organization	81
4.1.2	Organization Based on Primitive Functions	82
4.1.3	Structural Indexing and Organization	84
4.2	Retrieval of Design Analogues	85
4.2.1	Elaboration of Design Problems	86
4.2.2	Selection of Candidate Design Analogues	86
4.2.3	Ordering of Candidate Design Analogues	87
4.3	Storage of the Target Design Analogues	91
4.3.1	Learning of Indices to Target Design Analogues	92
4.3.1.1	Functional Indices	92
4.3.1.2	Model-Based Learning of Functional Indices	94
4.3.1.3	Structural Indices	96
4.3.1.4	Model-Based Learning of Structural Indices	97
V.	NON-LOCAL MODIFICATIONS AND CROSS-DOMAIN TRANSFER:	
	USE OF GTMS	99
5.1	Spawning of Adaptation Goals	101
5.2	Multiple Adaptation Strategies	104
5.3	Adaptation by Instantiation of GTMs	107
5.3.1	Retrieval of GTMs	109
5.3.2	Instantiation of GTMs	112
5.3.3	Composition of Behaviors	113
VI.	PROBLEM REFORMULATION: USE OF GPPS	115
6.1	Internal Evaluation and Design Failures	116
6.2	External Evaluation and Design Failures	118
6.2.1	Knowledge Acquisition: External Feedback on Design Failures	121
6.3	Understanding Design Failures by Forming Causal Explanations	122
6.3.1	Retrieval of GPPs	122
6.3.2	Instantiation of GPPs	123
6.4	Problem Reformulation and Redesign	124
6.4.1	Discovery of New Constraints	124
6.4.2	Designing for the New Problem	124
6.4.3	Composition of Behaviors	126

VII. LEARNING DESIGN PATTERNS	129
7.1 Issues in Learning by Abstraction	129
7.2 Learning of GTMs	130
7.2.1 An Illustrative Learning Task: Learning of Cascading GTM	131
7.2.2 Different Interaction Conditions and Learning Situations	134
7.2.3 The Model-Based Learning Method	134
7.2.4 Incremental Revision of the Learned Mechanisms	139
7.2.5 Learning of Feedback GTM	143
7.3 Learning of GPPs	149
7.3.1 An Illustrative Learning Task: Learning of GPP of Heat Flow	149
7.3.1.1 Learning to Different Levels of Abstraction: Heat Exchange and Heat Flow GPPs	152
7.3.2 The Model-Based Learning Methods	153
VIII. LEARNING DEVICE MODELS	157
8.1 Learning by Model Revision and Primitive-Behavior Composition	158
8.1.1 When Oracle presents only the Desired Design upon Problem-Solving Failure	158
8.1.2 When Oracle presents only the Solution to the Specific Adaptation Goal upon Problem-Solving Failure	162
8.2 Learning by Model Revision	164
8.2.1 When there are no Problem-Solving Failures	164
IX. EVALUATION AND ANALYSIS	169
9.1 Evaluation of the Theory as a Whole	171
9.1.1 Computational Feasibility	171
9.1.2 Generality	171
9.1.2.1 Coverage of Different Types of Design Adaptation	172
9.1.2.2 Coverage of Different Tasks in Analogy	172
9.1.2.3 Coverage of Different Types of Analogies	173
9.1.2.4 Coverage of Different Domains	173
9.1.3 Common Representations	173
9.1.3.1 For Different Types of Models	173
9.1.3.2 Across Different Processes in Analogical Design	173
9.1.4 Scalability	174
9.2 Evaluation of the Individual Components of the Theory	174
9.2.1 Problem Solving	174
9.2.1.1 Coverage of Different Design Tasks	175
9.2.1.2 Comparison of Different Adaptation Strategies for Efficiency	175
9.2.1.3 Quality of Solutions	177
9.2.1.4 Use of the Problem Solver for A Completely Different Task	178
9.2.2 Learning of Design Patterns	178
9.2.2.1 Coverage of Different Types of Design Patterns	178
9.2.2.2 Expanded Coverage of Design Problems	179
9.2.2.3 Different Interaction Conditions in Knowledge Acquisition from External Feedback	181
9.2.2.4 Interactions of Learning with Other Components of the Theory	185
9.2.2.4.1 Influence of Design Analogue Selection on Learning	186
9.2.2.4.2 Influence of the Order of Presentation of Design Situations on Learning	188
9.2.2.4.3 Learning from Different Sets of Specific Design Situations	188
9.2.2.4.4 Learning from Design Examples with Different Structural Configurations	189
9.2.2.4.5 Influence of the Representation of SBF models on Learning	190
9.2.2.4.6 Influence of the Internal Organization of SBF models on Learning	190
9.2.3 Qualitative Modeling and Learning of Device Models	193

9.2.3.1	Coverage of Different Types of Models	193
9.2.3.2	Different Methods of Acquiring Device Models	193
9.2.3.3	Use of the Same Representations for a Completely Different Task	194
9.2.4	Memory: Organization, Indexing, and Index Learning	194
9.2.4.1	Efficiency of Retrieval for Different Indexing and Organization Schemes	194
9.2.4.2	Use of the Indexing Scheme for Retrieval in a Completely Different Task	197
9.3	Limitations	197
9.3.1	Limitations Of Components of the Theory	197
9.3.1.1	Problem Solving	197
9.3.1.2	Learning of Design Patterns	198
9.3.1.3	Qualitative Modeling and Learning of Device Models	199
9.3.1.4	Memory: Organization, Indexing, and Index Learning	200
9.3.2	Limitations Of the Theory as a Whole	200
9.3.2.1	Coverage of Non-Local Modifications	201
9.3.2.2	Coverage of Cross-Domain Transfer	202
9.3.2.3	Coverage of Problem Reformulations	202
X.	RELATED WORK	203
10.1	Qualitative Modeling	203
10.1.1	Content and Representation of Device Models	204
10.1.2	Content and Representation of Generic Models	206
10.1.3	Indexing and Organization	208
10.1.4	Acquisition	209
10.2	Design	211
10.2.1	Adaptive Design	211
10.2.2	Analogical Design	213
10.2.3	Innovation and Creativity in Design	214
10.3	Learning	216
10.3.1	Index Learning	216
10.3.2	Learning Abstractions from Experience	218
10.3.3	Learning of Strategies	219
10.3.4	A Model-Based Approach to Learning	221
10.3.5	Multistrategy Learning	222
10.3.6	Multistrategy Reasoning	223
10.3.7	Learning by Discovery	224
10.4	Analogical Reasoning	225
10.4.1	Purpose-Directed Analogy	226
10.4.2	Model-Based Analogy	227
10.4.3	Case-Based Approach	229
10.4.4	Psychological Theories	231
XI.	CONCLUSIONS	239
11.1	Issues Addressed	240
11.2	Contributions	240
11.3	Model-Based Analogical Design	248

LIST OF TABLES

2.1	Different Constituents in A Design Analogue	34
7.1	Situations of Regularity Between Similar Behavior Segments in Two Design Analogues	138
9.1	Individual Components of Our Theory and The Dimensions of Evaluation for Each	170
9.2	The Types of Non-Local Modifications that our Theory can Account for and its Limitations	201
10.1	How each of the above models of analogical transfer (including IDEAL which is a computational model) answers the critical theoretical questions	235

LIST OF FIGURES

1.1	A Computational Process for Model-Based Analogy in the Context of Design	12
1.2	An SBF Model of the Generic Physical Process of Heat Flow	14
1.3	An SBF Model of the Cascading GTM	16
1.4	A Design for A 1.5-volt Electric Circuit (EC1.5)	17
1.5	An Illustration of the Learning and Use of Cascading GTM in Adaptive Design	21
1.6	Design Analogue of Nitric Acid Cooler that Cools Nitric Acid from T_1 to T_2	22
1.7	A Partial Model of the Cascading GTM	24
1.8	Design Analogue of Nitric Acid Cooler that Cools Nitric Acid from T_1 to T_2	25
1.9	Functional Specification of A 3-volt Electric Circuit (EC3)	26
1.10	Behavior of the 3-volt Electric Circuit (EC3)	26
2.1	A Computational Process for Model-Based Analogical Design	32
2.2	An Illustration of the Process of Model-Based Analogical Design via Learning and Use of A GTM that We Call Feedback	45
2.3	Desired Function of A New Electronic Circuit	46
2.4	Function "Amplify Electricity" of A Simple Amplifier	47
2.5	Behavior "Amplify Electricity" of A Simple Amplifier	48
2.6	Behavior "Amplify Electricity" of the Desired Design	50
2.7	SBF Model of the Feedback GTM Learned from the Designs of the Simple Amplifier & the New Device	51
2.8	Desired Function of the New Gyroscope Control System	52
2.9	Function "Transfer Angular Momentum" of A Design Analogue (a simple Gyroscope Control System)	52
2.10	Behavior "Transfer Angular Momentum" of the Design Analogue (a simple Gyroscope Control System)	53
2.11	The subfunction formed by the Instantiation of the Feedback GTM	55
2.12	Behavior of the New Design Achieved by Composing the Behaviors of the Design Analogue (i.e., simple Gyroscope Control System) and the Subdesign (i.e., worm)	56
3.1	Sulfuric Acid Cooler	59
3.2	The Structure Schema for the Sulfuric Acid Cooler	59
3.3	Structure Schema	60
3.4	Function and Behavior of Sulfuric Acid Cooler	62
3.5	Functional Specification Schema	63
3.6	Behavioral State Schema	64
3.7	Behavioral State Transition Schema	66
3.8	Primitive Function (and Behavior) ALLOW	70
3.9	Representation of Substances: Examples	71
3.10	A Partial Memory of Substances in IDEAL	72
3.11	Representation of Components: Examples	73
3.12	A Partial Memory of Components in IDEAL	73
3.13	An SBF Model of a Heat Exchanger	75
3.14	An SBF Model of the Heat-Flow GPP (i.e., The Zeroth Law of Thermodynamics)	77

3.15 A Complete Description of the Cascading Mechanism in SBF Representation	79
4.1 A snapshot of a functionally organized analogue memory	82
4.2 A Snapshot of A Memory Organized by Primitive Functions	83
4.3 Snapshots of IDEAL's structurally organized analogue memory	84
4.4 Function of Cooling High-Acidity Sulfuric Acid	86
4.5 The Selection Algorithm	88
4.6 Design of Low-Acidity Nitric Acid Cooler	89
4.7 Design of High-Acidity Sulfuric Acid Cooler	93
4.8 A model-based method to obtain functional indices for design analogues	95
4.9 Snapshots of IDEAL's functionally organized analogue memory	96
4.10 A model-based method to obtain structural indices for design analogues	98
5.1 Illustration of The Relationship Between Two Types of Transfer Processes in IDEAL	101
5.2 IDEAL's Method for Spawning Adaptation Goals	103
5.3 IDEAL's Partial Process of Analogical Design via GTMs	107
5.4 The SBF Representation of A Feedback Mechanism	109
5.5 The Two Designs of Gyroscope Follow-up <i>before</i> and <i>after</i> instantiating the Feedback Mechanism	111
5.6 IDEAL's Method for Instantiating A GTM	114
6.1 IDEAL's Method for Model Revision	117
6.2 IDEAL's Partial Process (Evaluation and Redesign) of Analogical Design	119
6.3 Failed and Redesigned Coffee Makers	120
6.4 Function and Behavior of An Initial Design of Coffee Maker	121
6.5 A Failure of Coffee Maker	122
6.6 The GPP of Heat Flow	123
6.7 A Causal Explanation for One Failure of Coffee Maker	124
6.8 A New Function (subfunction) Desired of Coffee Maker	124
6.9 Function and Behavior of A Design Analogue of Hot-Plate	125
6.10 A new sub-behavior of Coffee Maker that achieves the new subfunction	126
6.11 A Fragment of the Behavior of The Redesigned Coffee Maker	127
7.1 Task of Learning Generic Teleological Mechanisms from Design Analogues	131
7.2 Design of A 1.5-volt Electric Circuit (EC1.5)	132
7.3 Design of A 3-volt Electric Circuit (EC3)	133
7.4 A model-based method for abstracting GTMs over regularities in design analogues	135
7.5 A Few Patterns of Regularity in Device Behaviors	136
7.6 A Complete Description of the Cascading Mechanism in SBF Representation	140
7.7 Design of A 4-volt Electric Circuit (EC4)	142
7.8 An Alternative Design of the 4-volt Electric Circuit (alternative with respect to the configuration of batteries)	143
7.9 Learning of Feedback Mechanism	145
7.10 A Complete SBF Model of the Feedback Mechanism	148
7.11 Task of Learning Generic Physical Processes from Analogues	149
7.12 Sulfuric Acid Cooler	150
7.13 Function and Behavior of Sulfuric Acid Cooler	151
7.14 Function and Behavior of Nitric Acid Cooler	152
7.15 A model-based method for abstracting over parameter transformation	153
7.16 A model-based method for abstracting over location transformation	154
7.17 Behavior of a Heat Exchanger (i.e., the GPP of Heat Exchange) abstracted from SAC and NAC	155

7.18	The Zeroth Law of Thermodynamics (i.e., the GPP of Heat Flow)	156
8.1	Structure of the 3-volt Electric Circuit in Schema Form	159
8.2	Incomplete & Incoherent Behaviors of the 3-volt Electric Circuit in the middle of the process of generating behavior from structure	160
8.3	Behavior of Battery ₁ in the context of the 3-volt Electric Circuit	161
8.4	Complete Behaviors of the 3-volt Electric Circuit generated from the given structure	162
8.5	Illustration of Generating the Structure of EC3 from EC1.5 and the New Sub-Structure (all structures shown only diagrammatically)	164
8.6	Evolving Design for EC6 with repeated Applications of Component-Addition Strategy on the Design of EC1.5	167
8.7	Evolving Behavior for EC6 with repeated Applications of Component-Addition Strategy on the Behavior of EC1.5	168
9.1	The Class of Problems IDEAL Can Solve BEFORE It Learns the Cascading Mechanism	180
9.2	The Class of Problems IDEAL Can Solve AFTER It Learns One Type of Feedback Mechanism	182
9.3	The Class of Problems IDEAL Can Solve AFTER It Learns A Second Type of Feedback Mechanism	183
9.4	An SBF Representation of the Device-Composition Mechanism	187
9.5	An Alternative Structural Configuration for the Design of EC3	190
9.6	An Alternative Specification of the Behavior of EC1.5	191
9.7	Flatly Organized Behavior of the Design of A 3-volt Electric Circuit	192
9.8	Performance Measures on the Analogue Retrieval Task	196

SUMMARY

Design is a wide-ranging and open-ended information-processing task. A common method for generating designs is to adapt previously encountered designs. In this method, selected elements of a previous design are transferred to the new problem. Such adaptive processes are believed to play an important role in reasoning underlying innovation and creativity. But current theories of adaptive design are limited to routine design. That is, they cover only design problems with fixed problem specifications, local modifications to previous designs, and within-domain transfer of design knowledge.

The goal of this research is to build a theory of innovative design that addresses these limitations. We characterize innovation in design as including (1) non-local modifications to previous designs, (2) cross-domain transfer of design knowledge, and (3) reformulation of problem specifications. Non-local modifications are changes in the topology of a past design, i.e., changes in the arrangement of elements in a design. Non-local modifications are needed in adaptive design because past designs with the required topology (for solving new problems) may not always be available. Non-local modifications are hard because in some domains the design elements can be strongly interacting and making changes to the arrangement of the design elements can globally affect the elements in different parts of the design.

Cross-domain transfer of design knowledge is the transfer of experience gained in solving design problems in one domain (such as electric circuits) to solve a problem in another (such as heat exchangers). Cross-domain transfer is needed because the necessary designs may not always be available in the same domain as that of a given problem. Cross-domain analogies are hard to make because of the difficulty in recognizing the similarity between a target problem and a source analogue; the difficulty may also be in determining what knowledge to transfer from the source domain to the target and how to do the transfer. These tasks are hard because the source and target domains may refer to different design elements at a given level of abstraction.

Problem reformulation means revisions to problem specifications, such as addition, deletion, or modification of constraints, after the design process begins. It is generally needed because of the late arrival of constraints into problems and the discovery of constraints later in the design process. Problem reformulation is hard because it requires a designer to reason about the current design and its failures, and then to formulate new constraints if necessary.

To address the above issues, we investigate two related hypotheses. First, we hypothesize that high-level abstractions in device design enable non-local modifications to previous designs, cross-domain transfer of design knowledge, and reformulation of design problems upon evaluation of initial designs. Our theory is that *design patterns* are a particularly useful class of

abstractions that enable innovative device design. Design patterns capture different kinds of relationships, for instance, functional and causal relationships, among design elements, and they can be represented as generic, case-independent models. This research focuses on two particular types of design patterns: generic physical processes (e.g., heat-flow process) and generic teleological mechanisms (e.g., cascading and feedback mechanisms). The former captures causal relationships among design elements while the latter captures functional relationships. We study how design patterns enable non-local modifications, cross-domain transfer, and problem reformulation.

Second, we hypothesize that design patterns can be acquired from specific design experiences including feedback from an oracle upon problem-solving failure. Since the particular types of design patterns considered in this research capture functional and causal relationships, the design experiences from which those design patterns can be learned need to also capture those types of relationships among specific design elements. A Structure-Behavior-Function (SBF) model of a device explicitly specifies how the device works, that is, how the functions of structural elements get composed into the functions of the device as a whole. SBF models of specific designs thus enable learning of the functional- and causal-type design patterns and also provide a language for representing them as models.

The theory of innovative adaptive design based on design patterns also leads us to a general theory of *model-based analogy*. Model-based analogy integrates learning of useful abstractions, such as design patterns, with their use in analogical problem solving. A computational system called IDEAL (Integrated Design by Analogy and Learning) implements model-based analogy and evaluates it for device design. Detailed experiments with IDEAL, involving some 50 different designs from five different device domains verify and validate our hypotheses.

CHAPTER I

INTRODUCTION AND OVERVIEW

Let us consider the problem of designing a structure that enables people to climb a small hill. Depending on the knowledge available, a designer may solve this problem in different ways. If the designer has access to a past design that achieves a similar function, he can adapt the design to the new problem. In this example, the previous design is from the same domain as the new problem, the domain of hills. Suppose that the previous design is a "staircase" whose structure is composed of stair steps of one-foot high each. If the terrains of the current and the past hills (for which structures are needed to climb) are similar, then the required modification may be small, simple, and local. Examples of simple modifications are increasing or decreasing the number of stair steps and changing the height of each stair step. These local modifications change only parameters of design elements but not their arrangement. Such local modifications to the structure in a past design sometimes can result in a new structure that meets the requirement of new problem.

Suppose the terrains of the current and the past hills are different: the terrain of the past hill has a gentle slope, but the slope of the current hill is steep. Suppose that the stairs in the previous design are linear: the structure consists of a sequence of stair steps placed along a straight line from the bottom to the top of the elevation of the hill. The linear configuration of staircase does not solve the new problem because the slope of the current hill is steep. The issue becomes how to modify the past design so that the new design delivers the desired functionality and satisfies the constraint from the terrain. In this example, small, simple, and local modifications do not suffice. Instead, there is need for a non-local modification such as the modification of the linear staircase into a circular one. A circular staircase is a sequence of stair steps arranged in a manner that they spiral around a vertical axis. Non-local modifications, in general, are more innovative because they make changes to design topology, i.e., the arrangement of the design elements.

Now consider the situation in which the designer does not have access to a past design for climbing hills. But suppose that he has seen the design of a flight of stairs used in a house that enables people to go from one floor to another. The designer might potentially use this knowledge to address the current problem. One possibility is that the designer is reminded of the staircase design from the domain of houses. This raises the hard issue of cross-domain reminding. Another possibility is that the designer may have already abstracted generic design knowledge from the past design. The generic design knowledge may enable the transfer of the "strategic

concept” of staircase to the new problem. The designer may now solve the new problem by designing a staircase that enables people to climb the hill.

Irrespective of the specific way in which the design is generated, it may fail. For instance, the use of a staircase for climbing the hill might reveal the difficulty of taking large, heavy objects to the top of the hill. The generated design for climbing the hill did not consider this requirement because it was not specified initially. But the design’s evaluation may point out the failure of the design to support the initially unspecified constraint of taking heavy objects up the hill. In order to redesign the staircase to avoid the failure, the designer may form a causal explanation for the failure. The causal explanation may both reveal the unspecified constraint and help in the generation of a design. The designer may redesign the staircase into a ramp that satisfies both the original and the new requirements.

This is an example of innovative design. Although it is now common to use circular roads around steep hills for easy climbing, it surely was considered an innovative solution for the problem when it was originally designed.

1.1 Issues in Innovative Design

In the above design example, we saw three key high-level issues in innovative design:

1. non-local modifications to past designs, i.e., changes to design topology
2. cross-domain transfer, i.e., transfer of design knowledge between distant domains such as houses and hills, or electric circuits and mechanical controllers
3. reformulation of design problems, i.e., the revision of problem specifications during the design process

1.1.1 Research Problem

The goal of this research is to build a theory of innovation in design, where innovative design can be characterized as including one or more of the above three factors. We focus on

1. device design, in particular, design of physical devices such as electric circuits, heat exchangers, and mechanical controllers;
2. conceptual design which involves producing a high-level specification of a desired artifact given the functional and structural constraints on the artifact;
3. non-local modifications and cross-domain transfer.

1.1.2 Research Hypothesis

Our hypothesis is that high-level design abstractions in the form of *generic design patterns* enable non-local modifications, cross-domain transfer, and problem reformulation. A design pattern is a specific type of knowledge that encapsulates a specific kind of relationship among the design elements. One example of a design pattern is the replication of a design element. This pattern occurs in a linear staircase in which a stair step is replicated to achieve the needed height. In the example of designing a structure for climbing the hill, this pattern was used for making cross-domain transfer. Another example of a design pattern is the circular arrangement of design elements so that they form a spiral around a vertical axis. In the example of the structure for climbing a hill, this pattern was used for making non-local modification.

Functional and Causal Design Patterns

Design patterns are not specific to the domains of hills and houses. Instead, they occur in all design domains. In the domain of physical devices, we hypothesize that a specially useful class of design patterns capture functional and causal relationships among the design elements. For an illustration of the use of a functional design pattern, consider an example from the domain of electric circuits. Suppose that the problem is to design an electric circuit that produces light of 12 lumens. If the designer has access to a previously designed circuit that delivers light of 6 lumens, he can adapt the past design to solve the new problem. Suppose the past design contains a bulb, a battery, and a switch connected in series so that the intensity of light is directly proportional to the voltage of the battery. If the voltage of the battery in the past design is 1.5 volts, then the designer may decide to simply replace the 1.5-volt battery with a 3-volt battery. But suppose that a 3-volt battery is not available. Then the simple modification of replacing the 1.5-volt battery with a 3-volt battery is not feasible. The issue then becomes how to modify the past design to produce light of 12 lumens. Suppose the designer knows the generic design pattern of replication of structural elements of a given functionality to achieve a larger functionality of the same type. Then the designer may instantiate this pattern and replicate the 1.5-volt battery twice to obtain the needed voltage of 3 volts. The design pattern of replication (also called *cascading* in engineering) captures a specific kind of abstract relationship between the functions delivered and the causal structure of the design elements. This is the same pattern of replication used in the staircase example earlier. Indeed, the designer may acquire the design pattern from a different domain and use it in the domain of electric circuits.

1.2 Conceptual Framework

We will now elaborate on our research issues and hypotheses at three levels of description: conceptual framework, computational theory, and computer program. At the level of conceptual framework, we explore analogical reasoning in innovative design, where analogy is mediated by

high-level abstractions such as generic design patterns. At the level of computational theory, we describe a specific mechanism of analogical reasoning for device design that we call *model-based analogy*. Model-based analogy uses the knowledge of qualitative device models and generic design patterns. At the level of computer program, we study the theory of model-based analogy in several engineering domains.

Now, we begin our description of the conceptual framework along three dimensions: the task, the method, and the knowledge.

1.2.1 The Task: Design

Design is a very common, wide-ranging and open-ended information-processing task. Humans, laypersons as well as professionals, encounter design task in various complexities and perform it “everyday”—designing a picture, designing a symphony, designing a program and designing a device are only a few examples. In all these kinds of design activity, a design task can be routine or non-routine depending on the designer’s knowledge and the design product. More specifically, a design task can be routine or non-routine depending on how similar or different the new design is from known designs.

1.2.1.1 Conceptual Design

Design process generally involves several different phases (or stages), such as conceptual design, configuration design, design realization, and design optimization. We focus on conceptual design, the preliminary phase of design. In this phase, the goal is to produce a high-level, qualitative specification of a desired design.

The specification of a design problem may evolve during the design process. The conceptual design task generally takes as input a specification of desired artifact’s function(s) and gives as output a high-level specification of the artifact’s structure that achieves the desired function(s). Besides functional constraints, others such as constraints on structure and its realizability may also be specified.

1.2.1.2 Adaptive Design

Design tasks, in general, may be solved by different reasoning methods depending on the available knowledge. A common method for generating designs is to adapt previously encountered designs. In this method, a new design problem is solved by modifying an old design solution such that much of the old design’s structure is preserved. Broadly speaking, current theories of adaptive design are limited to routine design as follows:

1. Adaptations to previous designs are local and parametric such as the replacement of one design element in a past design by another.

2. The transfer of designs is within-domain where new problems and past designs are from the same domain (at a given level of description of the domain). An example of within-domain transfer is in designing a new staircase for a house by transferring knowledge from the design of a staircase in another house.
3. Problem specifications remain fixed during the design process.

1.2.1.3 Innovative Design

Within the context of adaptive design, we characterize innovative design in terms of how the new design is relative to known designs. Accordingly, a design task is innovative if the task involves modifications to a known design's elements and its topology. More generally, innovative design involves one or more of the following:

1. non-local modifications to previous designs
2. cross-domain transfer of design knowledge
3. reformulation of design problems

Non-Local Modifications

Modifications to designs can range as follows: (1) changes to the parameters of the design elements (while preserving the design elements and the topology of the design), (2) changes to design elements (while preserving the design topology), and (3) changes to the design topology itself. Non-local modifications are changes to the design topology. The topology of a design refers to the arrangement of the design elements, that is, the configuration of the connections among the elements. Changes to a design's topology include addition or deletion of design elements, and connecting them in a different way. An example of a non-local modification is in modifying the design of a linear staircase for a hill into a circular staircase for another hill.

Non-local modifications are generally needed in adaptive design for two reasons: past designs with the needed topology may not be available and thus simple, local modifications may not suffice; even when a design with the appropriate topology is available, specific design elements with the desired functions may not be available to replace elements in the known design.

Making non-local modifications is hard because they can globally affect different elements in all parts of the design. Computing the effects of non-local modifications to a design can become intractable if the number of design elements is large and the elements are strongly interacting. The major issue is what knowledge might enable the control of inferences in making non-local modifications.

Cross-Domain Transfer

A design domain may be characterized by the design elements available in it. Batteries, bulbs, and wires are examples of design elements in the domain of everyday, simple electric circuits.

They are different from design elements such as pumps and pipes available in the domain of heat exchangers. The notion of a “domain,” in general, is fuzzy because it depends on the level of abstraction at which primitive design elements of the domain are described. For instance, engineering devices may be considered as belonging to the same domain as computer programs if design elements are described at a high level of abstraction where both programs and (physical) devices are viewed as abstract devices. Due to the fuzzy notion of domains, there lies a continuum between what is within-domain transfer and what is cross-domain transfer. In that sense, the domain of electric circuits is considered distinct from the domain of mechanical controllers, but more so from the domain of computer programs. Cross-domain transfer of design knowledge is transferring the experience gained in solving design problems in one domain (such as electric circuits) to solve a problem in another (such as heat exchangers).

Cross-domain transfer of design knowledge is needed because the necessary designs may not always be available in the same domain as a given design problem. But cross-domain analogies are hard to make because of the difficulty in recognizing similarity between a target problem and a source analogue and the difficulty in determining what knowledge to transfer from the source to the target and how to do the transfer. These tasks are hard because at a given level of abstraction in the description of the source and target domains, the target problem and source analogues may refer to different kinds of design elements.

Problem Reformulation

We define problem reformulation as including addition, deletion, or modification of constraints in a given problem after the process of design begins. A constraint in a design problem is the specification of a value or a range of possible values for some design parameter desired in the new design. An example of a constraint from the problem of designing an electric circuit is specifying that the desired voltage is 3 volts. Another example is specifying that the height of a staircase needs to be same as that of the hill, which we saw in the scenario of designing a “staircase” for climbing a hill.

Problem reformulation, in general, is needed because of the late arrival of constraints into problems and the uncovering of constraints later in the design process. That is, first, constraints on design problems may not be completely specified at the beginning of the design process but instead may be specified by an external agent later in the process. Second, some new constraints may be discovered and added to a problem specification when its initial designs are evaluated, or some old constraints may be deleted or modified after the designs are evaluated. In this research, we focus on the latter type of problem reformulation. In the scenario of designing a staircase for climbing a hill, we saw an instance of the discovery of new constraints—it was discovered late in the process that the slope of the hill needs be considered in the design of staircase for the hill. Reformulating problems by discovering new constraints and redesigning to incorporate the new constraints can lead to innovative designs.

Problem reformulation due to the discovery of constraints is harder and more interesting than the one due to the late arrival of constraints because the former requires a designer to reason about the current design and its failures, and then to formulate new constraints if necessary. Such problem reformulation is needed because some constraints on a design are not uncovered until after the design is evaluated, for example, evaluated by its use in a real environment. The constraints that are generally discovered late in the process are about interactions of a design with its environment. The discovery of such constraints is late because the conditions of the environment may not be known completely at the beginning of the process. Even if the conditions were known, they might change from the time of initial problem specification to the time of design realization. Therefore, when the environments are thus dynamic, designs may fail, although they satisfy the constraints of initial problem specifications. This type of problem reformulation is hard because it requires the following: (a) evaluation of the design by realizing and using it in real environments, (b) understanding of external feedback on design failures, and (c) discovery of new constraints. Then the reasoning issues become what knowledge might enable the understanding of external feedback and the discovery of new constraints, and how it might enable the tasks.

1.2.2 The Method: Analogical Reasoning

Analogical reasoning plays an important role in human problem solving and learning (Gick and Holyoak, 1980, 1983; Gentner, 1983; Ross, 1984). Analogical reasoning is the process of retrieving knowledge of a familiar problem or situation (called *source analogue*) that is relevant to a given problem (called *target problem*) and transferring that knowledge to solve the given problem. In our characterization of innovative design, the ability to transfer knowledge across domains is one of the important aspects of innovation. Thus our theory of innovative design also leads to a theory of analogical reasoning in design.

Analogies, in general, can be of different types (Vosniadou and Ortony, 1989): (1) within-problem, (2) cross-problem but within-domain, and (3) cross-domain. As we noted before in Section 1.2.1.3, there can be a continuum between what is within-domain analogy and what is cross-domain analogy because boundaries of a domain are generally fuzzy and they depend on the level of abstraction at which design elements in the domain are described. Within-problem analogy involves transferring knowledge from one subproblem to another subproblem within the context of solving the same overall problem, e.g., designing a staircase by transferring the design of another staircase in the context of designing a structure for climbing between floors in a multi-storeyed house. Cross-problem but within-domain analogy (hereafter called within-domain analogy) involves transferring knowledge from one problem in a domain to another problem in the same domain, e.g., designing a staircase for climbing a hill by transferring the design of a staircase for climbing another hill. Many existing computational models of analogical reasoning, such as *case-based reasoning* (Kolodner, 1993), address this type of analogy. Cross-domain

analogy involves transferring knowledge from a problem in one domain to another problem in a different domain, e.g., designing a staircase for climbing a hill by transferring design knowledge from the domain of houses (not necessarily from the design of a specific staircase in a specific house). Cross-domain analogies are the hardest to make because of the issue of recognizing similarity between two problems from two different domains and transferring knowledge between them. Some of the important issues in analogical reasoning include the following:

- What might be the content and representation of source analogues?
- How can a target problem be specified?
- Given a target problem, how does the retrieval of a source analogue occur? What kind of features in the target problem determine the retrieval?
- Once a source analogue has been retrieved, how might knowledge from the source be transferred to the target problem? Whether the knowledge from the source analogue is directly mapped onto the target or some abstraction shared by the source and the target mediates the transfer process?
- Since transferring knowledge from a source analogue to a target problem often may not satisfy the requirements of the target problem completely, how might the transferred knowledge enable inferences needed to complete the solution to the target problem? How might the transferred source analogue be modified?
- How can a solution for a target problem be evaluated? That is, how can a solution be verified to determine if it satisfies the requirements of a target problem?
- What might be learned from a target problem that may be 'useful' for future problem solving? What might be abstracted from the target and source analogues? How might this abstraction occur? When does this abstraction occur: at problem-solving time or at analogue-storage time?
- How might a target analogue be stored for later use? Whether it is stored at all? If it is stored, how might it be indexed¹ in memory? How might its indices be acquired dynamically?
- How might the abstractions (if any were formulated from the source and target analogues) be stored in memory? How might they be indexed?

Our theory of model-based analogy (MBA) attempts to address all the above issues. In our theory, each design analogue is associated with a model of the solution in the analogue (i.e., a

¹Indexing of knowledge can be generally viewed as "labeling" individual pieces of knowledge with appropriate information. That enables access of "a" particular piece of knowledge when a new situation specifies information similar to the label of that particular piece of knowledge.

device model). The transfer of design analogues from a source domain to a target domain is mediated by higher-level abstractions over device models in the form of functional and causal patterns.

1.2.3 The Knowledge: Device Models and Design Patterns

At the level of conceptual framework, our theory of model-based analogy in design depends on the content of knowledge that supports reasoning tasks in applying analogy for design. Our hypothesis is that design analogues, models of specific design solutions in the analogues (i.e., device models), and abstractions over device models (i.e., design patterns) enable innovative design by analogy. In our theory, a design analogue consists of a design problem, its solution, and a device model. The model of a device specifies and explains how the parts of the device give rise to addressing a design problem. The knowledge of device models is important for the control of reasoning in innovative design because the models capture how behaviors of design elements mediate between their functions and structures. For instance, consider the staircase design we saw earlier. A model for that design may specify how functions of each design element, i.e., 1-foot stair-step, get composed into achieving the overall function of enabling people to climb the hill. We hypothesize that the useful design abstractions for innovative design are abstractions over models of specific designs that we call design patterns. (More details on design patterns will be given later in Section 1.3.3.1.)

1.2.4 Issues and Research Hypotheses

The above characterization of innovative design (as in Section 1.2.1.3) in terms of non-local modifications, cross-domain transfer, and problem reformulation raises the following important issues. We summarize our research hypotheses at the level of conceptual framework for each of the issues below.

1. **Content and Representation of Design Knowledge:** What design knowledge might enable these three aspects of innovation? That is, what might be the content of design knowledge and how might it be represented?

Hypothesis: High-level design abstractions enable innovative design, i.e., non-local modifications to previous designs, cross-domain transfer of design knowledge, and reformulation of design problems upon evaluation of designs. These design abstractions are abstractions over models of specific design analogues (or cases). The model of a design may be characterized by the kinds of relationships it captures about the elements of the design. In particular, we refer to a model that encapsulates functional, structural, and causal knowledge about design elements. We call such models of specific designs as device models and the abstractions over them as design patterns.

2. **Access and Organization of Design Knowledge:** How might such design knowledge (as in (1) above) be accessed/retrieved from memory when a new problem is given? What kind of organization and indexing might support better retrieval of relevant knowledge from memory? How might the design knowledge be related to other types of knowledge, if any?

Hypothesis: Design knowledge, we refer to, can be of two types: (a) design analogues (and associated device models) and (b) design patterns (i.e., particular types of abstractions over device models). Design analogues and the associated device models can be retrieved by matching on problem specifications while design patterns can be retrieved by matching on types of functional relationships between designs. Design analogues need to be indexed based on the tasks for which they are used and they need to be organized hierarchically along those indices for an efficient and effective retrieval. Further, design analogues need to be associated with their models and the conceptual knowledge of design elements because that facilitates evaluation and elaboration subtasks in design.

3. **Use of Design Knowledge:** How might the design knowledge in (1) be used for innovative design? That is, what might be the specific processes by which that knowledge enables the three aspects of innovative design?

Hypothesis: Since the useful design knowledge is abstract (i.e., abstractions over models of designs such as design patterns), its use in new problems is through the instantiation of design patterns in the context of new problems.

4. **Origin and Acquisition of Design Knowledge:** Where does such design knowledge (as in (1)) that enables innovative design come from? How might it be acquired?

Hypothesis: Useful design abstractions such as design patterns can be acquired from specific design experiences including feedback from an oracle upon problem-solving failure.

1.3 Computational Theory

At the level of computational theory, our hypotheses are refined and focused in all the three dimensions: the task, the method, and the knowledge.

1.3.1 The Task: Device Design

A device is an artifact that has both output functions and internal causal mechanisms that result in output functions. The device design task takes as input a specification of the functional and structural constraints on a desired device and gives as output a specification of the structure of the device that delivers the desired functions and satisfies the structural constraints. Innovation in device design is especially hard. Consider the two aspects of innovative design, namely, non-local modifications and cross-domain transfer. Making non-local modifications to a known

device design is hard because the structural elements of the device can be strongly interacting and modifications local to one part of the device can have non-local effects. The issue is how to control the reasoning about non-local effects in modifying a known device design. Similarly, cross-domain transfer is hard because of the difficulty in determining “what” to transfer between distant device domains. The issue becomes hard because at the level of specific structural elements used to describe domains, it may not be possible to see the relevance of a source design to the target problem.

1.3.2 The Method: Model-Based Analogy

Figure 1.1 shows the computational process we developed for model-based analogy in design.² This computational process includes the following different stages: (1) retrieval of a source analogue; (2) transfer and modification (i.e., adaptation) of relevant knowledge from a source analogue to the target problem—this is a multistrategy transfer mechanism in which specific adaptation goals determine whether simple adaptations are done or complex ones using design patterns are performed; (3) evaluation of a solution to the target problem—the computational process also accommodates understanding of external feedback on design failures and possible, subsequent reformulation of design problems; (4) learning from the source and target analogues by abstraction—the process also accommodates interaction with an oracle for acquiring the target solution in case of problem-solving failures; and finally (5) storage of the solution to the target problem and the learned knowledge (i.e., design patterns). This process covers not only cross-domain analogies but also within-domain analogies. This work builds on the model-based approach to case adaptation developed in (Goel, 1989) that covers within-domain analogies in device design. Goel used the theory of structure-behavior-function (SBF) models for exploring issues in within-domain adaptive design and for representing models of specific devices.

In the process of model-based analogy, solving some classes of problems may involve the retrieval and instantiation of design patterns. But, for some other classes of problems in which adaptation goals are different, the transfer process may not involve the use of design patterns; instead, it may involve only simple modifications to known designs. The two specific types of design patterns, namely, Generic Physical Processes (GPPs) and Generic Teleological Mechanisms (GTMs), enable different subtasks of analogical reasoning in the computational process shown (Figure 1.1).

1.3.3 The Knowledge: SBF Models of Devices and Design Patterns

We adopt the theory of SBF models (Goel, 1989) to represent device models and design patterns. This theory takes the component-substance view in which devices are composed of com-

²Our goal in this research was to develop a computational theory rather than a psychological theory. All our arguments are intended to be functional from a computational perspective. However, in the chapter on related work, we will relate some parts of our theory to psychological research on analogy and postulate psychological predictions from our computational theory.

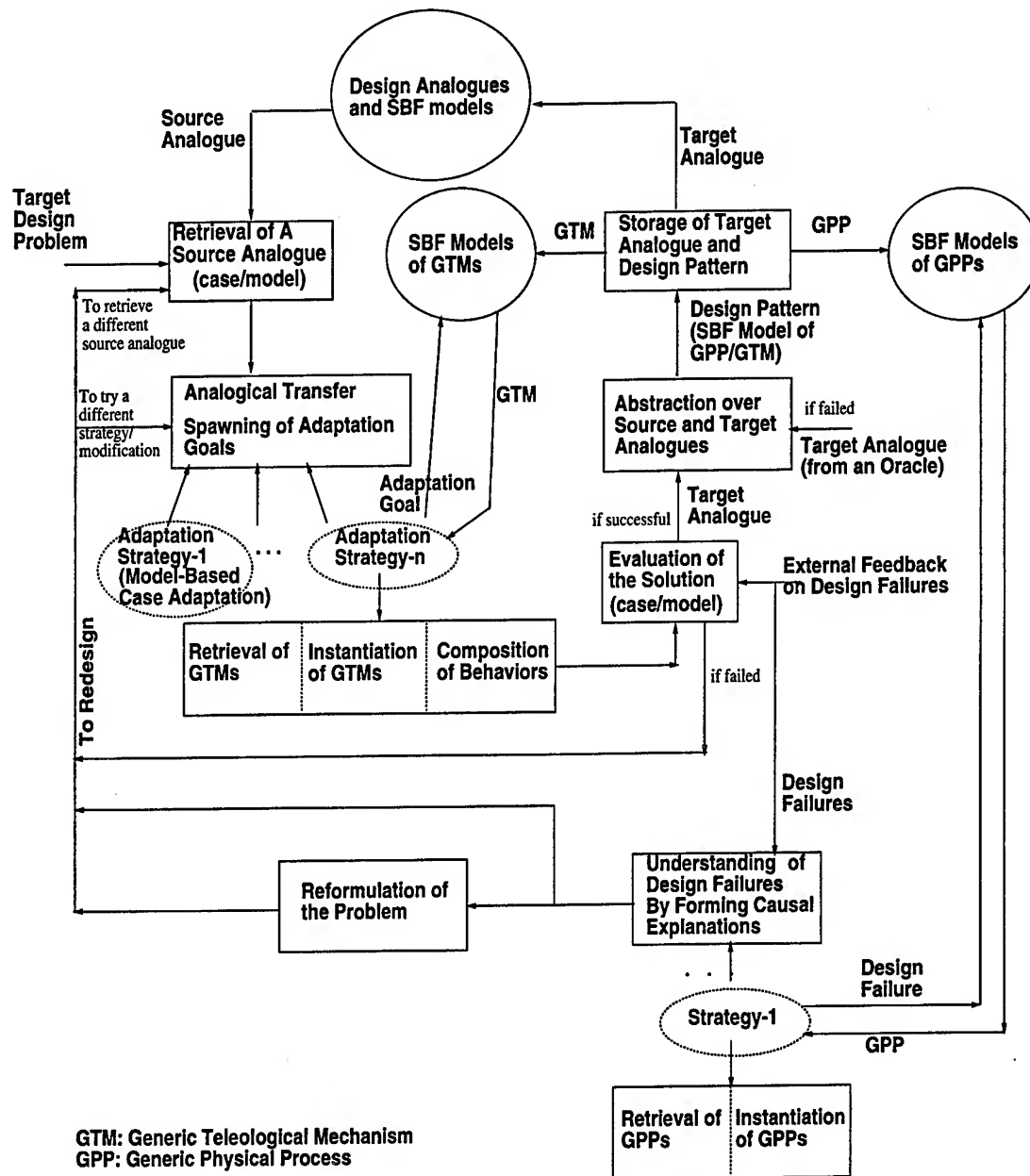


Figure 1.1: A Computational Process for Model-Based Analogy in the Context of Design

ponents and substances that flow among the components. Since design elements in that view are components and substances, the model of a device in that view specifies how functions of the components in the device get composed into achieving the device's functions. We will now elaborate on the structure-behavior-function (SBF) models of devices and of the two types of design patterns, which are the particular types of knowledge that enable innovative device design.

1.3.3.1 Design Patterns: GPPs and GTMs

Design patterns are a particularly useful class of abstractions that enable innovative device design. We hypothesize that in order to reason about devices with non-local, often global, effects in a tractable manner, modifications need to apply across collections of structural elements; otherwise, it is difficult to reason about non-local interactions among locally modified elements. In our theory, design patterns facilitate such controlled reasoning because they encapsulate the inferences needed to make non-local modifications in device design. Design patterns encapsulate relationships between functions and behavior of design elements, which, in turn, enable controlling the reasoning in innovative design. We also hypothesize that the cross-domain transfer of design knowledge needs to involve the use of high-level abstractions shared between the domains. More specifically, we hypothesize that design patterns abstracted from a source domain are “what” need to be transferred to solve problems in a target domain. Furthermore, the same kinds of high-level abstractions as above would be needed in order to deal with the hard issue of understanding device failures in the context of problem reformulation.

Different design patterns may capture different kinds of relationships among design elements, such as, spatial, temporal, functional, and causal relationships. Because of the kinds of relationships they capture, they can be represented as generic, case-independent models. Design patterns can be spatial patterns as in certain domains (e.g., design of pictures and buildings), or temporal patterns as in some others (e.g., design of symphonies and music), or functional and causal patterns as in yet other domains (e.g., design of physical devices such as electric circuits and heat exchangers). We focus on the functional- and causal-type design patterns, in particular, GPPs and GTMs.

A GPP is a causal-type design pattern which captures patterns of relations between the output and internal behaviors of physical devices. An example of a GPP is the process of heat flow—it specifies how heat flows from a hot body to a cold body when they are in contact. A GTM is both a functional- and causal-type design pattern which captures patterns of relations between functions (a subset of output behaviors) and internal behaviors of devices. An example of a GTM is a pattern which specifies how two or more devices with same functionality can be connected together to achieve a larger function—the mechanism is called “cascading.” A specific instance of the cascading GTM appears commonly in connecting two batteries together to get more voltage than the voltage each battery provides. Other examples of GTMs are feedback and feedforward mechanisms—which specify how to control variations in the output and the input

of devices respectively. A common device where feedback mechanism is used (i.e., instantiated) is the room heater with a temperature-setting knob; given a temperature setting, this device turns the heater “on” or “off” depending on the room temperature.

Design patterns can be viewed as *concepts* or *strategies* depending on the types of relationships the patterns capture. For instance, GPPs are concepts while GTMs are strategies. Since a GTM captures relationships between functions and internal causal behaviors of devices,³ it can be used to derive behaviors that achieve new functions, which is the task of design. Therefore, a GTM is a design strategy—more specifically, it is a design-adaptation strategy because it specifies how the behavior of an existing device needs to be modified to achieve new but similar function. On the other hand, GPPs capture the knowledge of physics concepts.

We represent GPPs and GTMs using the language of SBF models. The SBF language provides primitives to represent three types of knowledge about devices: physical structure of devices, functions of devices, and internal causal behaviors of devices that explain how the functions of structural elements get composed into the functions of overall structure. Since the design patterns capture only “patterned” relationships (i.e., types of relationships as opposed to specific ones) among design elements, their SBF representations are devoid of information about any specific physical structure of devices. However, those representations still capture the causal structure in the behaviors of devices. Hence, the function and behavior aspects of the SBF language are especially useful for representing GPPs and GTMs. Figures 1.2 & 1.3 respectively illustrate the SBF representations of the Heat-Flow GPP and the Cascading GTM. Note that

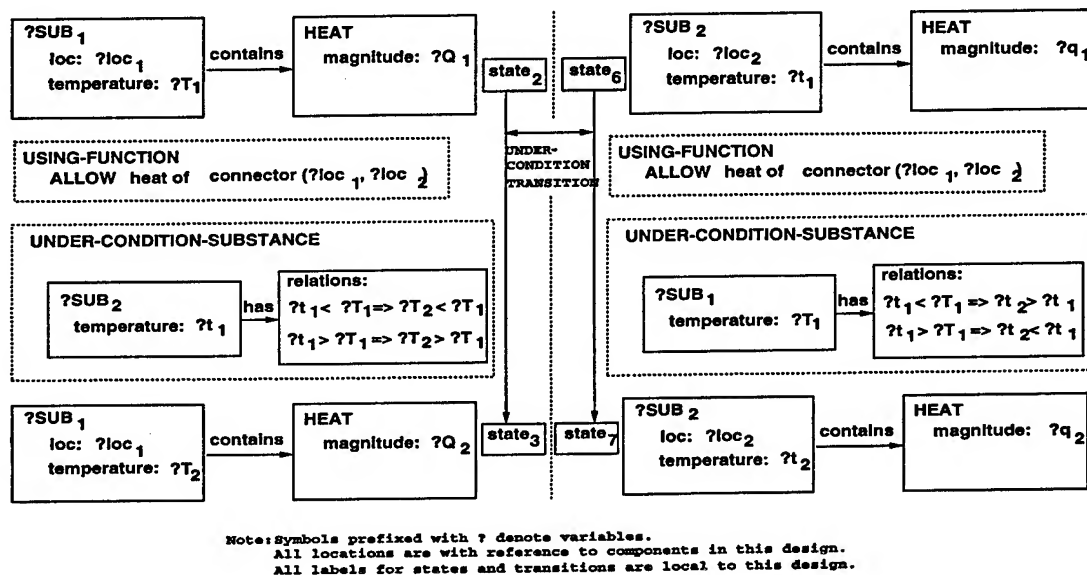


Figure 1.2: An SBF Model of the Generic Physical Process of Heat Flow

³More precisely, it captures mappings from the kinds of differences between functions to the kinds of differences between behaviors.

these do not refer to any specific substances or components that constitute the physical structure of any particular device. The SBF model of the Heat-Flow GPP shown in Figure 1.2 specifies how heat-flow occurs between any two substances when they are in thermal contact; it specifies how the temperatures of the two substances change in the heat-flow process. The SBF model of the Cascading GTM shown in Figure 1.3 specifies how a particular type of functional difference between any two designs maps onto particular types of relationships between the behaviors of those designs; it specifies how to modify the behavior of a candidate design in order to achieve a desired function.

1.3.3.2 Device Models and SBF Representations

Our hypotheses about the design patterns are based on a theory of device models. We defined a device model earlier in Section 1.2.3. That is, a device model specifies how causal relationships among specific structural elements in the device result in the device's functions. Because of the kinds of relationships in device models, we chose to represent them in the form of SBF models. The SBF model of a device captures the designer's comprehension of how the device works, that is, how the structure of its design results in its output behaviors (i.e., functions). As the name SBF model indicates, its constituents are the structure of a device (i.e., the physical structure), the functions delivered by the structure, and the internal causal behaviors that explain how the functions of structural elements get composed into the functions of the overall structure. An SBF model of a device thus organizes knowledge about the functioning of the device as a whole in terms of the functions of its elements.

We refer to the model of a device in a specific design analogue as a case-specific SBF model. In our theory, a design analogue not only encapsulates the problem (i.e., function(s) of the device) and its solution (i.e., the physical structure of the device) but also a model for the solution (i.e., the SBF model of the device). Figure 1.4, for example, shows a design for a simple electric circuit, EC1.5: Figure 1.4(a) presents a schematic of the device's structure; Figure 1.4(b) shows the device's function, which can be read as "given electricity with a voltage of 1.5 volts as input in the battery, the device produces light of intensity 6 lumens as output in the bulb when the switch is closed"; Figure 1.4(c) shows the device's causal behavior as a sequence of states and state-transitions that explains the internal causal process of the device. In SBF models, causal behaviors of devices can be represented at different levels of detail. For instance, Figure 1.4(d) shows the causal behavior of battery in the design at a more detailed level.

1.3.4 Refined Issues and Hypotheses

We now summarize the issues in innovative device design and our refined hypotheses at the level of computational theory.

1. **Content and Representation of Design Knowledge:** What knowledge of device design might enable the three aspects of innovation? That is, what might be its content

DESIRED DESIGN:

GIVEN:

?SUB ?prop1: ?val12

MAKES:

?SUB ?prop1: ?val22

BY-BEHAVIOR: Behavior B2

CANDIDATE DESIGN:

GIVEN:

?SUB ?prop1: ?val11

MAKES:

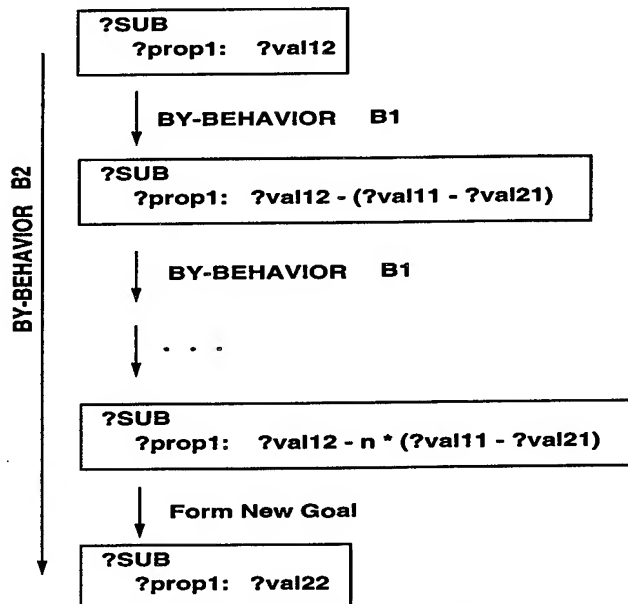
?SUB ?prop1: ?val21

BY-BEHAVIOR: Behavior B1

CONDITION:

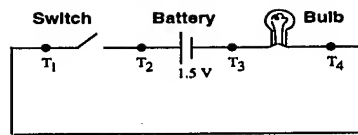
$?val22 - ?val12 \gg ?val21 - ?val11$

(a) Functional Difference the Cascading Mechanism Reduces

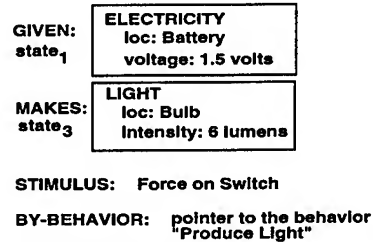


(b) Behavior Modification the Cascading Mechanism Suggests

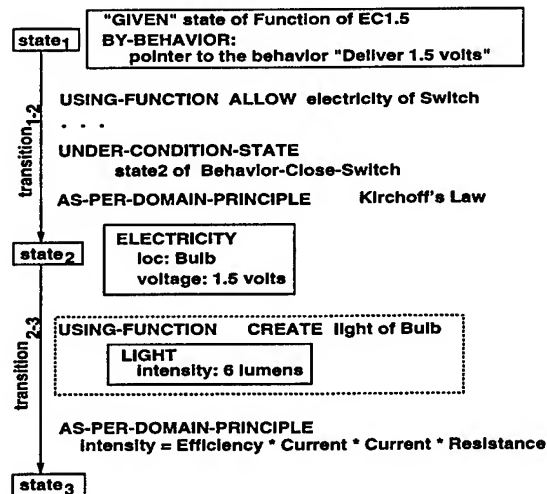
Figure 1.3: An SBF Model of the Cascading GTM



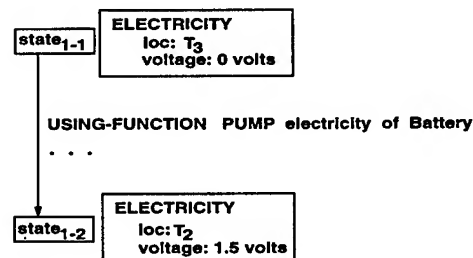
(a) 1.5-volt Electric Circuit (EC1.5)



(b) Function "Produce Light" of EC1.5



(c) Behavior "Produce Light" of EC1.5



(d) Behavior "Deliver 1.5 volts" of Battery

Note: All locations are with reference to components in this design. All labels for states and transitions are also local to this design.

Figure 1.4: A Design for A 1.5-volt Electric Circuit (EC1.5)

and how might it be represented?

Hypothesis: Useful high-level design abstractions of a particular type that we call design patterns enable innovative device design. Design patterns are useful because they encapsulate relationships between functions and behaviors of design elements, which, in turn, enable controlling the reasoning in innovative design. Two types of design patterns are specifically useful: GPPs and GTMs. GPPs capture patterns of relations between the output and internal behaviors of physical devices (e.g., heat-flow process). GTMs capture patterns of relations between the functions and internal behaviors of devices (e.g., cascading and feedback mechanisms). We adopt the theory of SBF models to represent not only the models of specific devices but also the GPP and GTM design patterns. Since the design patterns are abstractions over regularities in device models, the patterns can also be specified in terms of SBF models.

2. **Access and Organization of Design Knowledge:** How might such device-design knowledge (as in (1) above) be accessed/retrieved from memory when a new problem is given? What kind of organization and indexing might support better retrieval of relevant knowledge from memory? How might the design knowledge be related to other types of knowledge, if any?

Hypothesis: Since in device design, analogues are designs of specific devices and design problems consist of specifying different kinds of constraints, such as functional and structural, the analogues need to be indexed by the respective types of knowledge about devices. In our theory, any piece of knowledge should be indexed depending on the tasks for which that might be used and the nature of relationships that knowledge might capture. Designs of devices need to be hierarchically organized along functional and structural indices for an efficient and effective retrieval. We hypothesize that using the indices based on the vocabulary of SBF models enables an efficient and effective retrieval. Device designs can be accessed by matching on the specification of functions and structural constraints in a design problem respectively with the functions and structures in the problems of stored designs. Design patterns such as GPPs and GTMs do not refer to specific structural elements in devices. Since GPPs capture causal relationships and we use them for the task of understanding design failures in the context of problem reformulation, the GPPs are indexed by behavioral abstractions over causal behaviors in the GPPs. Similarly, since GTMs capture functional and causal relationships and are used as strategies for achieving design-adaptation goals, they are indexed by types of functional differences that the GTMs specify how to reduce.

3. **Use of Design Knowledge:** How might the design knowledge (in (1)) be used for innovative device design? That is, what might be the specific processes by which that knowledge enables the three aspects of innovation in device design?

Hypothesis: The use of design patterns in adaptive design is by their instantiation in the context of specific designs and problems. Such process enables innovative device design. When design patterns are instantiated in specific design analogues, they can enable non-local modifications in a tractable manner. When design patterns are acquired in one domain and problems are given in another, the design patterns can enable cross-domain transfer by being retrieved and instantiated in the new problem domain. Further, design patterns can enable problem reformulation by facilitating the formation of causal explanations for device failures. In particular, we explored how GPPs enable problem reformulation, how GTMs enable non-local modifications to known designs, and how both enable cross-domain transfer.

4. **Origin and Acquisition of Design Knowledge:** Where does such design knowledge that enables innovative device design come from? How might it be acquired?

Hypothesis: Design patterns can be acquired from specific design experiences by abstracting over the SBF models of specific devices in those experiences. This process may include feedback from an oracle in case of problem-solving failures. SBF models of specific designs enable learning of design patterns by abstraction over patterns of regularities in the designs, and provide a language for representing the design patterns as generic models. The SBF models of devices may themselves be acquired in different ways: (1) by revision of known models of similar devices (Goel, 1991b), (2) by a combination of model revision and composition of behaviors of primitive structural elements, and (3) by instantiation of design patterns in the models of known devices.

1.4 The Computer Program: IDEAL

The most refined level of our theory is the level of the computer program that instantiates the theory. We implemented the proposed theory of model-based analogy in a system called IDEAL⁴ in the context of innovative design of physical devices. In addition to supporting the “computational feasibility” test of our theory, IDEAL serves as an experimental testbed to evaluate the theory in several other dimensions that are described in a later section. IDEAL performs both within-domain analogies and cross-domain analogies in designing physical devices. It performs innovative design, i.e., non-local modifications to known designs, cross-domain transfer of design knowledge, and problem reformulation. In IDEAL, we implemented all stages of model-based analogy and integrated learning, memory, design problem solving, and comprehension of devices. IDEAL evaluates our theory in multiple domains such as electric circuits, heat exchangers, simple electronic circuits, coffee makers, and mechanical controllers. Furthermore, it demonstrates how the specific types of design patterns, namely, functional- and causal-type design patterns (e.g., GPPs and GTMs), enable innovative design.

⁴IDEAL stands for Integrated “DEsign by Analogy” and Learning.

IDEAL evolved from another system called KRITIK (Goel, 1989) which implements a model-based approach to making within-domain analogies for designing physical devices. Thus IDEAL inherits most of its capabilities of within-domain analogy from KRITIK. The most important and novel aspects of IDEAL are (1) its abilities to learn abstractions from design analogues in one domain and transfer them to solve design problems in other domains and (2) its ability to make non-local modifications to known designs when necessary.

1.4.1 An Illustrative Example

We will now illustrate how design patterns enable innovation in design with the example of a particular design pattern, namely, cascading GTM, from IDEAL. Cascading means connecting multiple devices of same functionality together to get a similar but large function. For instance, a common way of obtaining a 6-volt supply is by connecting four 1.5-volt batteries together that involves applying the cascading mechanism.

Figure 1.5 presents the complete story of our illustration. This story has two parts: as shown in the top half of the figure, one part concerns the learning of the cascading GTM, and as shown in the bottom half of the figure, the other part concerns the use of the cascading GTM in design adaptation. In this illustration, we first focus on how the cascading GTM enables non-local modifications to a known design in the domain of heat exchangers, and then present how the cascading GTM can be learned from specific designs in the domain of electric circuits. The specific examples of design problems and the GTM are from the computer program IDEAL. That is, in the following description, IDEAL is the designer solving the problems and learning the GTM.

Suppose now that a designer is given a problem which specifies the desired function as one of transforming the temperature of Nitric Acid from T_1 at one end of NAC-Device (an input location) to T_2' ($T_2' < T_1$) at the other end of the device (an output location). Suppose also that the design of a Nitric Acid cooler (see Figure 1.5 bottom-left) that changes the temperature of Nitric Acid from T_1 to T_2 ($T_2 < T_1$; $T_2' < T_2$) is available in the designer's analogue memory and that the designer retrieves it. Figure 1.6 further shows the function and the partial behavior of the available design of Nitric Acid cooler. In model-based approach, a designer uses the model of a known design to determine which components (or substructures) in the design can be modified in order to achieve the desired function. In this example, suppose that the designer selects the entire device of Nitric Acid cooler to modify (because the device's model does not decompose the device behavior into behaviors of individual components). Consider that the designer's knowledge is in a state where no single component (or structure) available delivers the desired functionality. Then, under such knowledge state, simple modifications to a known design of Nitric Acid cooler such as replacing a substructure (e.g., heat-exchange chamber or the entire NAC-Device) by another will not result in a device that can solve the given problem because the required range of cooling is much more than T_1-T_2 . It is in such problem-solving

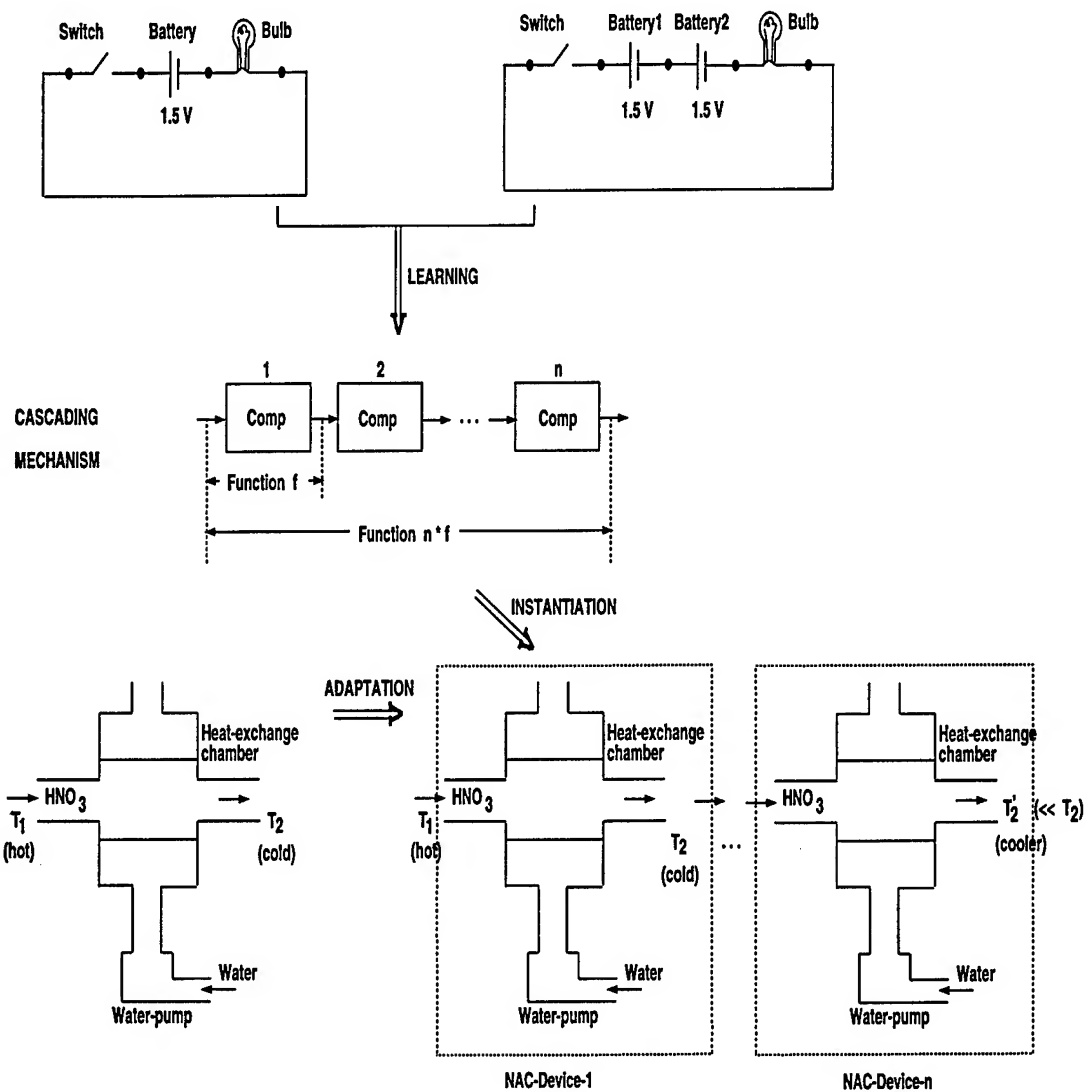


Figure 1.5: An Illustration of the Learning and Use of Cascading GTM in Adaptive Design

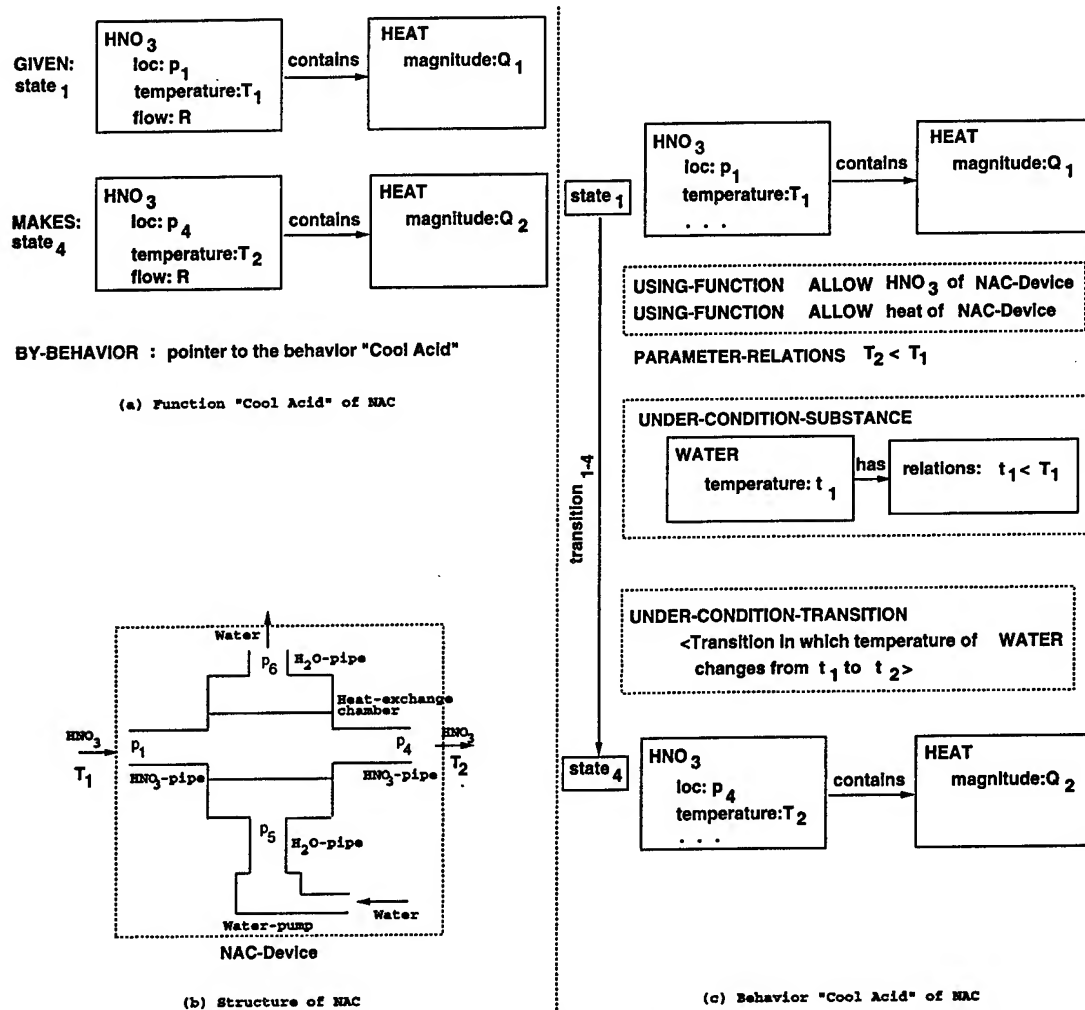


Figure 1.6: Design Analogue of Nitric Acid Cooler that Cools Nitric Acid from T_1 to T_2

situations that the knowledge of design patterns can enable a designer to perform non-local, patterned modifications to the known design and thus solve the current problem.

Suppose for a moment that the designer knows of the cascading GTM (shown schematically in Figure 1.5 center). Figure 1.7 further illustrates the designer's model of the cascading GTM in SBF representation.⁵ The cascading mechanism encapsulates *when* and *how* to use multiple devices with same, small functions in order to achieve a large function. If the designer can recognize that the functional difference between the known design and the new design problem is "similar" to the functional difference that the cascading mechanism reduces, it can use the cascading mechanism in the current context and thus solve the problem. The solution obtained by retrieving and instantiating the cascading mechanism in the context of the current design (i.e., the entire NAC-Device) is shown in Figure 1.5(bottom-right). Figure 1.8 further shows the behavior resulting from instantiating the cascading GTM in the behavior of the known NAC device. Thus the use of the cascading GTM can result in innovative design by enabling topological, non-local modifications to known designs.

Let us now present the second part of our story in which we illustrate how a design pattern may be learned from design experiences. In particular, we describe how the designer may have learned the above cascading GTM from designs of electric circuits. Suppose first that the design of a 1.5-volt electric circuit (EC1.5) whose schematic is shown in Figure 1.5(top-left) is available in memory. Figure 1.4 shows the function and behavior of the design also. Consider now the scenario where the designer is presented with a problem of designing an electric circuit that delivers the function illustrated in Figure 1.9. The SBF representation in Figure 1.9 specifies that the desired function is to produce light of intensity 12 lumens as output in bulb, given electricity with a voltage of 3 volts as input in battery. Suppose an additional constraint is specified: there is no battery that can supply electricity with a voltage of 3 volts but there are batteries that can deliver electricity of 1.5 volts.

The designer can use the desired function as a probe into its memory of past designs and retrieve the design analogue EC1.5 because the given functional specification is similar to the function of EC1.5. Because there are differences in the desired function and the function of the retrieved design, the designer tries to adapt the retrieved design next. But if the designer knows only of simple modifications, then it may fail. (Note that the designer does not have the knowledge of the cascading GTM at this point.) The designer may then fail to solve the problem due to the additional structural constraint specified. Then, if the designer is given external feedback, such as a correct design that solves the new problem, it can learn adaptation knowledge (i.e., a strategy) that would be useful in problem-solving situations like the current one.

A solution to the given problem above is a circuit in which two 1.5-volt batteries are *cascaded*

⁵Note that this is a partial model of the cascading GTM compared to one shown in Figure 1.3. This representation specifies the behavior modification needed when the transformation in a desired function is an integral multiple of that in a candidate function, i.e., n is an integer.

DESIRED DESIGN:

GIVEN:

?SUB ?prop1: ?val12

MAKES:

?SUB ?prop1: ?val22

BY-BEHAVIOR: Behavior B2

CANDIDATE DESIGN:

GIVEN:

?SUB ?prop1: ?val11

MAKES:

?SUB ?prop1: ?val21

BY-BEHAVIOR: Behavior B1

CONDITION:

$?val22 \sim ?val12 \gg ?val21 \sim ?val11$

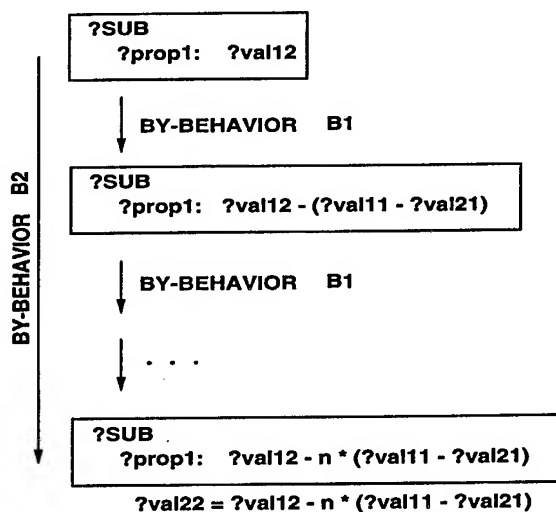
(a) Functional Difference the Cascading Mechanism Reduces**(b) Behavior Modification the Cascading Mechanism Suggests**

Figure 1.7: A Partial Model of the Cascading GTM

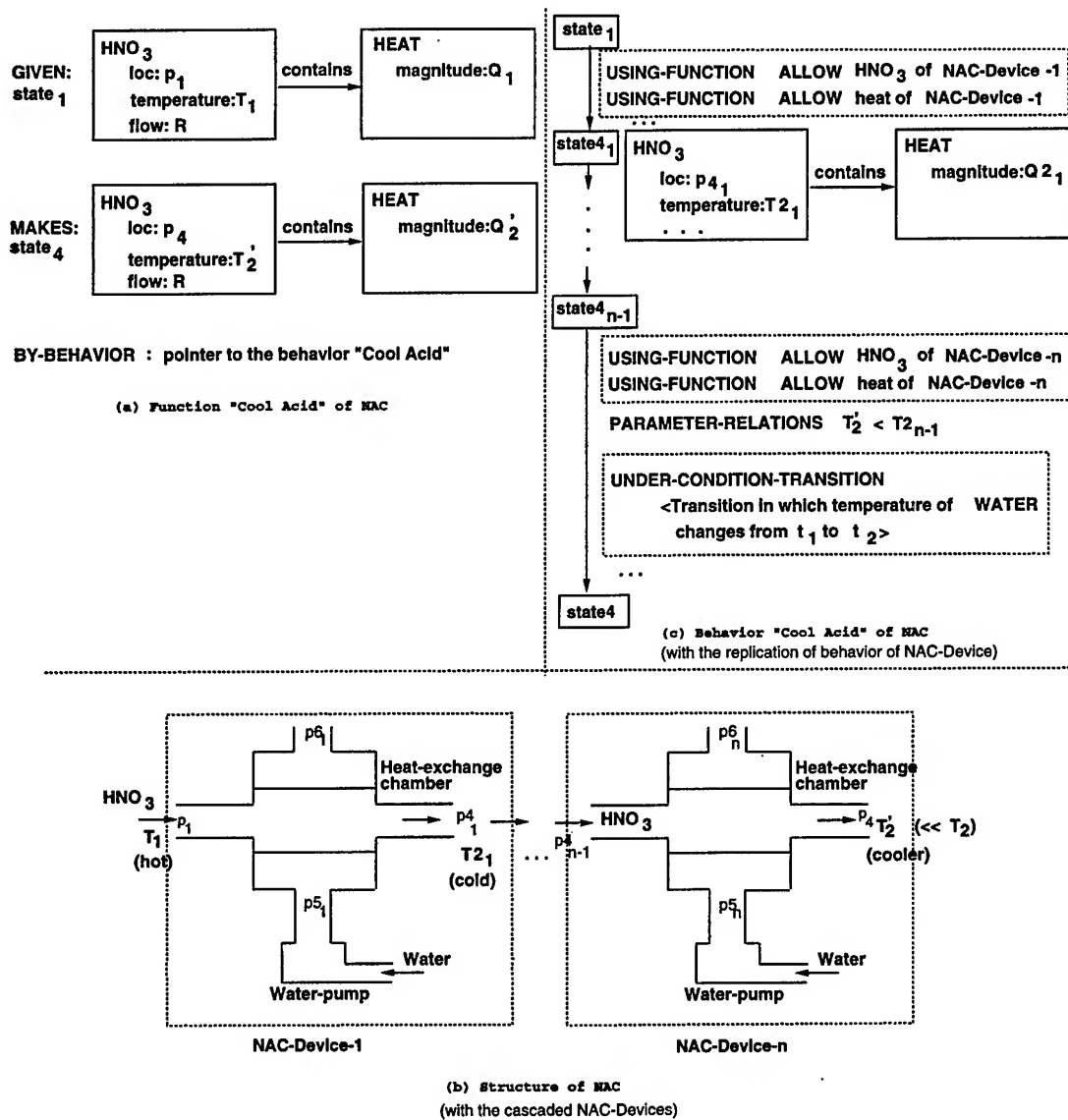


Figure 1.8: Design Analogue of Nitric Acid Cooler that Cools Nitric Acid from T_1 to T_2'

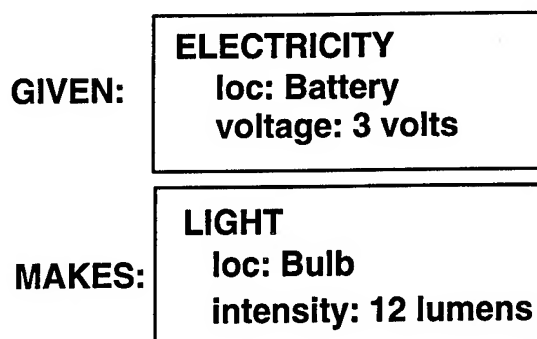


Figure 1.9: Functional Specification of A 3-volt Electric Circuit (EC3)

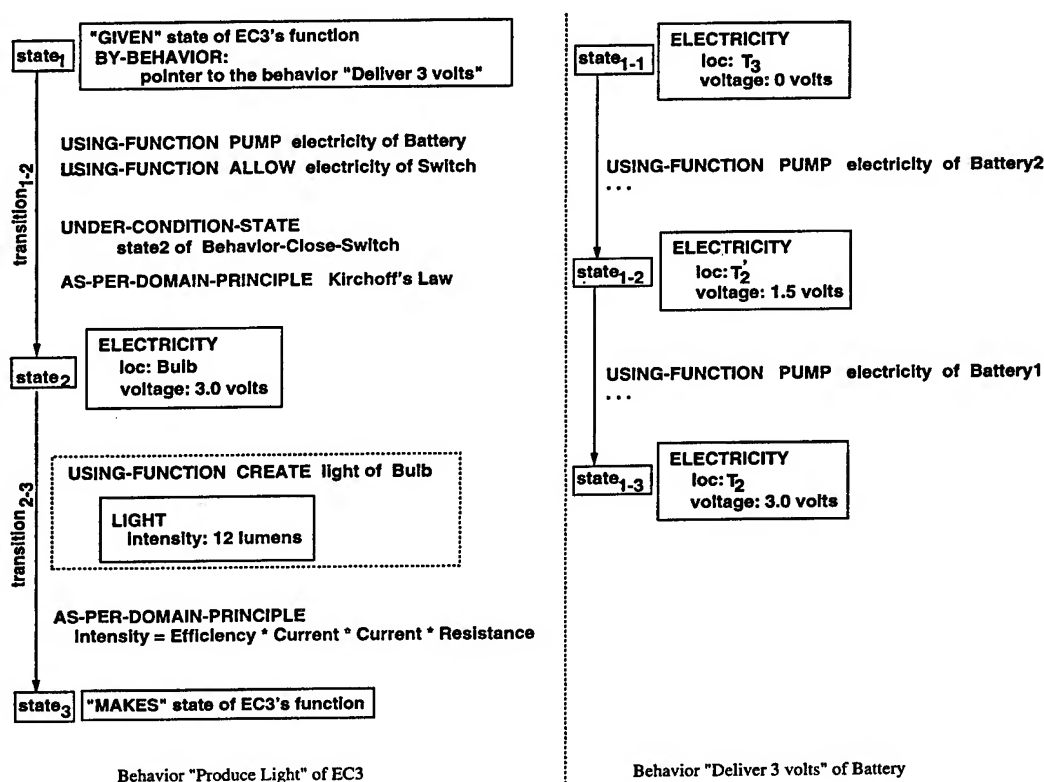


Figure 1.10: Behavior of the 3-volt Electric Circuit (EC3)

(see Figure 1.5 top-right). Given the correct structure of the new device, the designer can first learn how the device behaves (Figure 1.10) by revising the behavior of EC1.5, and then also learn a new strategy for adaptation (which we call *cascading GTM*).⁶ An SBF representation

⁶Based on the two design examples, one with a single battery and the other with two batteries, the designer can hypothesize a schema for replicating n components by comparing the models of the designs and abstracting over regularity in the designs.

of the learned (more precisely, *hypothesized*) cascading mechanism and its index are shown in Figure 1.7; the functional difference that the cascading mechanism reduces is the index for the mechanism.⁷ The description in Figure 1.7 is a partial model of the cascading mechanism. A more complete description for cascading would indicate that a behavior can be replicated as much as possible to achieve a desired function and then a goal can be formed to achieve any remaining function; such a residual function will be needed when the desired function is not an integral multiple of the function of each replicated segment. The complete description was shown in Figure 1.3 in this chapter.

The designer can revise the hypothesized (partial) description of the cascading mechanism into a more complete description when it encounters a new design problem whose solution has an instance of the complete cascading mechanism. Suppose for instance that the designer later encounters a design problem with a functional specification of producing light of intensity 16 lumens as output in bulb when the switch is closed, given electricity with a voltage of 4 volts in battery as input. In addition, suppose that there is a structural constraint: there is no battery that can deliver electricity with a voltage of 4 volts but there are batteries that can deliver electricity of 1.5 volts and one battery of 1 volt. A solution to this problem will have a physical structure where two 1.5-volt batteries and one 1-volt battery are cascaded. The designer can potentially revise the mechanism based on a design. Thus acquiring a complete model of the cascading mechanism may involve solving a number of design problems incrementally. We will revisit these examples and the learning process in more detail in later chapters.

As illustrated in the above story, design patterns that enable non-local modifications to known designs in one domain may be learned from design experiences in another. Thus design patterns can also enable cross-domain analogical transfer and thereby enable innovative design.

1.5 Methodology and Evaluation

In general, addressing issues in a complex phenomenon such as innovative device design raises a number of questions. Our hypothesis that high-level design abstractions enable innovative device design raised many that we listed in the previous sections. Most of those complex questions often do not have unique answers. Therefore, in addressing such issues, we take a “design science” view (Simon, 1969). The idea is “not to commit” to a single answer (e.g., a particular architecture, a particular processing, or a particular content theory) but rather to explore different possible answers and to identify conditions under which those answers might be plausible and identify the behaviors that they result into. For instance, in this research we explored different interaction conditions in acquiring design patterns from design experiences and external feedback rather than committed to a particular type of interaction.

⁷When a new piece of knowledge such as the cascading GTM is learned, its usefulness relies on the ability to also learn appropriate conditions under which it might be used. In other words, learning a piece of knowledge inevitably involves learning *indices* for the knowledge.

We believe that problem solving, learning, memory, and comprehension need to be closely integrated. For instance, learning of knowledge cannot be separated from the problem-solving tasks for which the learned knowledge might be used. It is also methodologically useful to consider several aspects of intelligence, such as problem solving and learning, together in building computational theories because theories of each aspect will be *behaviorally* constrained by those of the others. That is, the ways in which different abilities interact impose constraints on each ability. For example, learning can be constrained by the problem-solving task for which the product of learning might be used. Specifically, the problem-solving task can act as an evaluation of learning and can guide the selection of right indices for the learned knowledge.

Our general research methodology is an iterative process: propose a computational theory for the chosen task(s); implement the theory in a computer program; evaluate the theory by running several types of experiments (see below); revise the computational theory and then go back to the first step. For instance, our initial theory of learning GTMs was too specific to learning of the cascading mechanism and it failed to meet the evaluation criterion of generality (of learning different types of generic mechanisms). We then revised (more precisely, generalized and expanded) our theory so that it can also account for several other generic mechanisms such as feedback, feedforward, and device composition.

Evaluation

In this research, we used the following dimensions of evaluation to test the theory as implemented in the computational system IDEAL.

1. **Computational feasibility:** We implemented the proposed theory in IDEAL and it works in the problem domain(s) considered.
2. **Generality:** There are five different aspects to the criterion of generality.
 - (a) **Coverage of Different Design Tasks:** The larger the coverage of specific problems or tasks in the domain(s) of consideration the better accepted and more general is the theory. We tested the proposed computational theory on a “representative” set of problems (i.e., different classes of problems) in the domains considered.
 - (b) **Coverage of Different Types of Design Adaptation:** We have ensured that the theory is general enough to account for different types of design adaptation, for instance, adaptive design and redesign based on external feedback from an evaluation of designs.
 - (c) **Coverage of Different Tasks in Analogy:** This means generality of the theory in that it covers multiple tasks in analogical reasoning. That is, we have ensured that our theory covers issues in multiple stages of analogical reasoning such as retrieval, transfer and modification, evaluation, learning, and storage. It is important to consider these different issues because solutions to each impose constraints on those for the others; otherwise, the resulting theories may be underconstrained.

- (d) **Coverage of Different Types of Analogies:** A theory of analogy is more general if it covers multiple types of analogies, for instance, within-domain and cross-domain analogies. Our theory covers both types of analogies.
- (e) **Coverage of Different Domains:** This is to ensure that the theory is applicable in multiple domains. We tested the proposed computational theory on problems from different domains. This evaluation was also necessary to show cross-domain analogies.

A major issue here is "how many different things" are sufficient to support any generality and whether the needed domain knowledge is easily accessible to the experimenter for testing. In the particular domains and tasks this research considered, it's been very hard to infuse the necessary knowledge into the system and conduct the evaluations.

3. **Different Interaction Conditions in Knowledge Acquisition from External Feedback:** This is to ensure that the theory is general enough to account for different interaction conditions with an oracle that provides external feedback to the designer when problem-solving failures occur. We ensured that our theory covers multiple instances of the task of learning GTMs where the input information varies across different instances.
4. **Common Representations:** This is to establish the adequacy of same representations, for instance, SBF models, in supporting a number of different processes and tasks such as generation of designs, evaluation of designs, and learning of different types of design patterns.
 - (a) **For Different Types of Models:** In our theory, the same SBF language was sufficient to represent both device models and design patterns.
 - (b) **Across Different Processes in Analogical Reasoning:** In our theory, same representations were used successfully to support inferences across different processes in analogy.

5. Computational properties:

- (a) **Efficiency:** This criterion ensures that the proposed theory is computationally tractable and efficient. That is, for instance, the system that implements the theory solves problems in reasonable number of steps and reasonable time. In developing methods for various tasks, we made sure that they are efficient. We tested our theory of indexing designs for efficiency on retrieval task and compared alternative schemes of indexing and organization.
- (b) **Scalability:** This is to ensure that the theory can account for complexity in individual designs and also do so for a number of different designs and other types of

knowledge. We tested our representations and methods for designs of moderate complexity and size (i.e., for instance, designs consisting of orders of 10 design elements and 10 structural relations).

- (c) **Comparison of Different Adaptation Strategies:** This is to test the competence and the performance of alternative adaptation strategies for solving different design tasks and to identify conditions under which a given strategy may be more appropriate to use. Comparing the performance of different strategies is meaningful only if the strategies enable solving a common class of design tasks. In our theory, only two strategies (i.e., component addition and cascading GTM) satisfy these conditions and we compared them for efficiency at problem solving.

1.6 Organization of the Thesis

Chapter 2 describes the computational process of model-based analogy and its underlying theory, and gives a global picture of different subtasks in the process and the control of processing. It also introduces the different types of knowledge in the process and their content that enables the process(es).

Chapters 3-8 present the technical details of this work. Chapter 3 describes the content theory of device models and of GPP and GTM design patterns. Chapter 4 presents the indexing and organization of design analogues in memory, and describes how the analogues can be retrieved. This chapter also describes how different types of indices to new design analogues can be automatically learned and how the memory can be dynamically reorganized while storing the new analogues. Chapter 5 describes cross-domain analogical transfer in design and presents the model-based method that makes use of device models and design patterns. In particular, this chapter describes the use of GTMs in adapting source design analogues. Chapter 6 describes how new designs may be evaluated internally and externally, and how design failures can be explained using design patterns, in particular, GPPs. Chapters 7 & 8 describe the analogical learning tasks addressed in this research and the model-based methods proposed for them. Specifically, these learning tasks involve learning the two types of design patterns from design experiences and learning new device models. The former chapter covers learning of design patterns while the latter covers learning of device models.

Chapter 9 describes our experimental evaluation of model-based analogy and a detailed analysis of some subtasks and alternative ways of solving them. It also presents the limitations of our current theory and sets pointers to further work. Chapter 10 compares this work with related research. Finally, Chapter 11 concludes the thesis with contributions of this work to several specific issues in the research problem.

CHAPTER II

A COMPUTATIONAL THEORY OF MODEL-BASED ANALOGY

In this chapter, we will present a detailed account of our computational theory of model-based analogy (MBA). Figure 2.1 reproduces the computational process of model-based analogical design from Chapter 1. In MBA, the task of analogical design is decomposed into the following subtasks; each of these subtasks has a corresponding process (or set of processes) as illustrated in Figure 2.1.

1. Retrieval of a source analogue: this includes the elaboration of the target problem description, the recognition of relevant source analogues, and the selection of the best matching analogue.
2. Transfer and modification: this includes the identification of “what” knowledge in the source analogue to transfer to the target problem, the localization of differences between the source and the target to be fixed (i.e., spawning of adaptation goals—we also refer to this as diagnosis) and the elimination of those differences (i.e., achieving the adaptation goals by applying an adaptation strategy—we also refer to this task as repair), and completion of the solution (in case, the source analogue does not completely solve the target problem, a solution may be achieved by posing subproblems, solving for them, and composing their solutions).
3. Evaluation of the solution: this involves the verification of the proposed solution for the target problem by simulating a model for the solution or by implementing and executing the solution in the real world; when the solution fails in either type of evaluation, it may be redesigned by understanding failures if necessary; if a solution cannot be generated at all, then an oracle can interact with the MBA process and provide the process with different kinds of feedback on the solution which enables the next subtask in MBA.
4. Abstraction over the source and target analogues: this involves learning an abstraction (in particular, a design pattern) general enough to explain some aspects in both the source and target analogues; this also involves learning indices for the learned design pattern.
5. Storage of the solution (i.e., the target analogue) and the learned design pattern: this involves learning proper indices for the analogue and placing the analogue and the design pattern in memory in the “right” place.

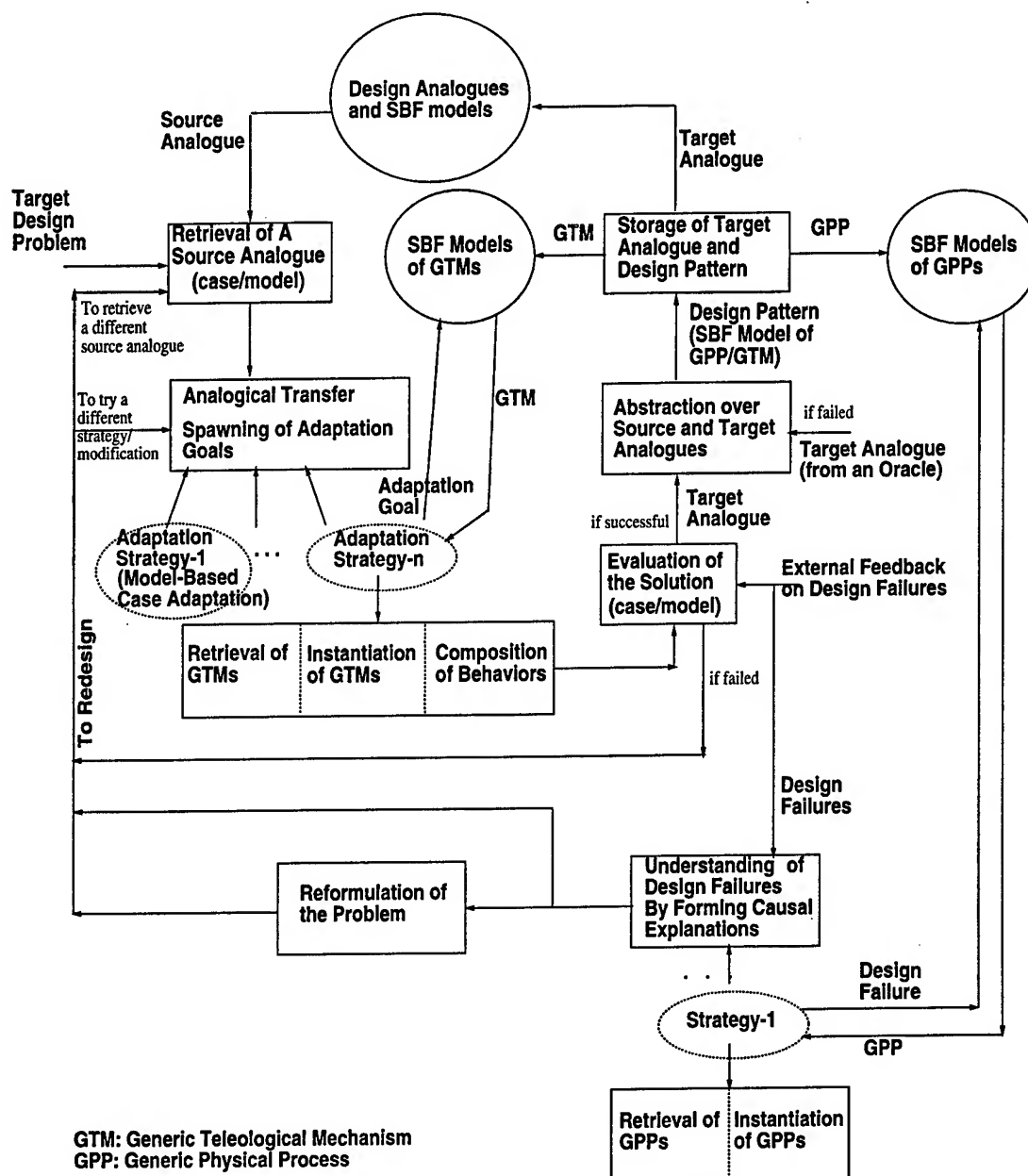


Figure 2.1: A Computational Process for Model-Based Analogical Design

We will now consider each of the important aspects of MBA and describe the control of processing. First we will describe the task and the knowledge, and then describe each subtask in MBA. In the current implementation of the MBA process in IDEAL, each type of design pattern enables different aspects of innovative design. As shown in Figure 2.1, GTMs enable the subtask of achieving adaptation goals and GPPs enable the subtask of understanding design failures.

2.1 The Conceptual Design Task

We developed the MBA theory in the context of the conceptual design of physical devices. The conceptual design phase of the design task generally takes as input a specification of different types of constraints on a desired artifact, such as the functions desired of the artifact and the structural constraints on the artifact. It involves producing as output a high-level specification of the structure of the desired artifact. Therefore, in our theory, the design task takes as input a specification of the functional and structural constraints on a desired design (i.e., the target design problem), and gives as output a structure (i.e., the solution) that realizes the specified function and satisfies the structural constraints. However, in real world, problem specifications may not remain fixed during the design process; hence in MBA, the problems can be reformulated after the design process begins. Problem reformulation may be triggered by a number of different sources including the evaluation of candidate designs. (We refer to a potential solution for a target problem as a candidate design and the solution itself as a target design or desired design.) Our theory accounts for problem reformulation based on the evaluation of designs. Since a model of the designed artifact is an important source of knowledge for the model-based process, in our theory, the design task gives as output not only the structure of a new design but also an SBF model that explains how the structure realizes the desired function.

2.2 Design Analogues, Device Models, and Design Patterns

As shown in Table 2.1, a design analogue may consist of different constituents: a design problem, a design solution, a model of how the solution satisfies the requirements of the problem (i.e., a device model), an outcome of evaluating the solution, and a reasoning trace or a hint of the method(s) used. In principle, a design analogue may consist of any combination of these five constituents. In this research, we explore one combination in which a design analogue consists of a design problem, its solution, and a model of the solution (Goel, 1992). This combination is highlighted in Table 2.1.

In device design, a typical problem consists of the specification of the functions desired of the device and a solution consists of the specification of the structure of the device that achieves the desired functions. In some domains (such as engineering domains), a comprehension or model of how the structure of a device delivers its desired functions is often available. When such

Table 2.1: Different Constituents in A Design Analogue

No.	Constituent	Its Description
1	Design problem	A specification of different types of constraints, such as functional and structural, on the desired artifact.
2	Design solution	A specification of the structural elements and their composition that solves the design problem.
3	Device model	A model of the design solution, i.e., how the structure of the design achieves the desired function and satisfies the structural constraints in the problem.
4	Outcome	A specification of whether (and perhaps how) the solution succeeded or failed when implemented/simulated in an environment.
5	Reasoning trace	A specification of the decisions made along the design process and their justifications.

knowledge is available, a design analogue can also consist of the model of the device. These models are designer's (human or machine) "mental models" of how the devices work. This research adopts the structure-behavior-function (SBF) representations (Goel, 1989) to describe the models of devices. Hence, a design analogue in MBA specifies (1) the functions delivered by a known design, (2) the structure of the design, and (3) an SBF model of the design. As we stated in Chapter 1, indexing is "labeling" individual pieces of knowledge with appropriate information for enabling their retrieval (efficiently and effectively) in new situations. In MBA, design analogues are indexed both by functions that the stored designs deliver and by the structural constraints they satisfy because the problem specifications consist of them. In indexing, an important issue is to determine what vocabulary (i.e., terms) to use for labels of knowledge. Note that labels need not be single terms such as "design-1," instead they can be knowledge structures themselves such as functional specifications. In our theory, the language of SBF models provides the indexing vocabulary for both design analogues and design patterns in MBA. The design analogues are organized along multiple dimensions of generalization where the dimensions pertain to the constituents of design functions and structural constraints. Such an organization is chosen in order to enable efficient retrieval based on partial match between the target problem and the source analogues. A source analogue is considered to be a partial match to the target problem if only some information in the index of the source is same as the information specified in the target. If everything in the index of the source is same as the information in the problem, then that source analogue is an exact match.

Besides design analogues and device models, design patterns are also important source of knowledge in MBA. Design patterns are abstractions over functional and behavioral relationships in device models. Our hypotheses for different issues in analogy are based on a theory of design patterns and device models. Making cross-domain analogies requires abstractions that can be retrieved and instantiated in the context of the target problem. In design by model-based analogy, design patterns enable cross-domain analogies and other interesting issues in design such as non-local modifications and problem reformulation. Our theory, in particular, covers the content, representation, learning and use of the particular types of design patterns, which are generic teleological mechanisms (GTMs) and generic physical processes (GPPs). Hence, Figure 2.1 refers to only these two types of design patterns. We adopt and extend the SBF language for representing the design patterns; we chose the SBF language because of the specific types of relationships captured in the design patterns.

2.3 The Subtasks of Model-Based Analogy

We will now trace through the different paths of control in the process of MBA illustrated in Figure 2.1.

2.3.1 Retrieval of A Source Analogue

The first task in the MBA process is to retrieve similar analogues given a new problem. The main issue here is how to decide on the similarity of an analogue to the given problem. Addressing this issue requires a language for specifying the new problem. It also requires a vocabulary for indexing the analogues. In our theory, SBF models provide such a language. Thus in the MBA process new problems are specified in SBF language. Given a target design problem, the process of MBA first uses the problem as a probe into the analogue memory and retrieves a matching source design analogue (and the SBF model associated with the source design). The MBA process tries to retrieve as specific a design analogue as possible by searching through the multiple hierarchies along those dimensions that are specified in the problem. If a single perfectly matching analogue (i.e., an exact match) is retrieved, then the target problem is solved by simply transferring the solution from the retrieved analogue to the new problem without any modification. But often multiple design analogues match the given problem partially. Then there is a need for selecting "an" analogue from the retrieved. In MBA, the best matching analogue is selected because we believe that the solution from an analogue that matches the best with a given problem would require the least modification to become a solution to the given problem.

The issue then becomes how to determine the best matching analogue. One method for selecting the best matching analogue is to compute differences between the target problem and the problems of the retrieved analogues and order them by the differences. Then the best matching analogue is the retrieved analogue that is least different from the target problem. Comparing the differences with one another requires a uniform language to express those differences. In MBA, the SBF language provides the vocabulary to capture the differences between the target and source problems. But, the SBF language by itself does not provide enough guidance on the issue of ordering the differences. Therefore, MBA uses qualitative heuristics to address that issue and to select an analogue.

Computing Differences between the Source and Target Problems: Since the SBF language provides the vocabulary for representing a variety of design problems, and it is a well-defined and uniform language across different problems, computing the differences between problems in MBA is not computationally complex. Computing the differences between two problems simply involves comparing the features of the functions in the problems and classifying the differences in the values of the features into known types of functional differences. Examples of the types of functional differences due to the SBF language are substance difference, substance-property-value difference, and component difference (Goel, 1989). The differences between problems computed for the purpose of ordering the retrieved analogues will also be useful in the next subtask of MBA, which is transfer and modification. In general, some differences between two problems may be more important than others and it may be useful to determine

the relative importance of differences. In our theory of MBA, this issue is not yet dealt with. But we addressed in MBA how multiple differences between two problems can be ordered among themselves (as described below) for the purpose of transfer and modification.

2.3.2 Transfer and Modification

This subtask is activated when the source analogue matches the given problem partially. This involves first setting up the adaptation goals given the differences between the source design problem and the target problem, and then achieving those goals. On the other hand, when the source analogue is an exact match to the target problem, this task is trivial: it only involves copying the solution and the model from the source analogue to the target problem and no adaptation of the solution from the source analogue is needed. Hence we will only discuss this task in case of a partially matching source analogue for the target problem.

2.3.2.1 Spawning of Adaptation Goals

When the functions specified in the target problem and the source analogue are different, MBA spawns the goal of adapting the partial design solution in the source analogue. The issue is how to form adaptation goals given the differences in the source and target problems. Addressing this issue requires a typology of differences that can be mapped to a typology of adaptation goals. Since in MBA the SBF language provides a typology of functional differences, the same typology can be viewed as indicative of adaptation goals. Different types of functional differences lead to different types of adaptation goals. Some functional differences may give rise to multiple goals to achieve. In these cases, MBA uses the heuristic of achieving simpler goals first and more complex ones later. The question then becomes how to determine which goals are simpler and which are complex. For addressing this question, the MBA process uses the same qualitative heuristics as it uses for ordering functional differences. As mentioned in Chapter 1, some functional differences in some domains may give rise to goals that require global changes to source designs. Depending on the complexity of the source design, making global changes can be computationally complex. Therefore, in order to keep the adaptation computationally tractable, it is desirable to localize the differences to smaller substructures in the source design as much as possible. Forming localized adaptation goals requires knowledge that decomposes the design not only structurally, but also both functionally and behaviorally. Such knowledge is necessary because the task is a complex task of mapping from the differences in the problems (i.e., the functions) onto the structure to be modified. SBF models exactly satisfy these knowledge needs. Hence the process of MBA uses the SBF model of the source design and the overall functional differences, and traces through the model to identify all possible, local differences to be reduced (i.e., basically the process of diagnosis). In solving certain classes of design problems under specific knowledge conditions (i.e., when specific types of designs are available and when models are incomplete), however, it may not be possible to localize the differences and hence the goals. In those situations,

to control the processing complexity in making non-local adaptations, MBA uses the knowledge of design patterns because design patterns encapsulate the inferences needed for making non-local modifications.

2.3.2.2 Multistrategy Adaptation

In general, different adaptation strategies are applicable for achieving different adaptation goals. Some goals may be achieved by strategies that make simple modifications (such as parameter tweaks) and some others may require strategies that make more complex modifications (such as topological changes). The need for some of the complex, non-local modifications may arise from the structural constraints which specify what components can be used in (or are available for) the new design. As shown in Figure 2.1, the process of MBA has multiple adaptation strategies including model-based case adaptation. Some of these strategies are simple ones such as component replacement and some are more complex ones such as instantiation of GTMs. Given an adaptation goal, the process of MBA checks in a fixed order if the known adaptation strategies achieve the goal. The known strategies are assumed to be preordered from simpler ones to more complex ones. The typology due to the SBF language is not sufficient to enable the MBA process determine which strategies are simpler and which are complex. Thus in MBA, simple modifications to the source design are attempted first, and only if the simpler ones do not apply to the current adaptation goals or do not lead to a target solution, then more complex ones are considered.

Some adaptation strategies use the knowledge of GTMs in making complex, non-local modifications that involve changes to the device topology in the source analogue. The issue then is how to access GTMs given the adaptation goals. In the MBA process, GTMs are indexed by the patterned functional differences for which they suggest patterned modifications, and the adaptation goals are expressed in terms of specific functional differences to be reduced. The choice of the indexing scheme is also influenced by the specific task for which the GTMs are used in MBA. The MBA process uses the adaptation goal as a probe into its memory of GTMs in order to retrieve a GTM and instantiates the retrieved GTM in the context of the target problem. To use a retrieved GTM, it is necessary to check if its instantiation violates any given structural constraints (i.e., to check if the needed structural elements are available in memory). If the retrieved GTM satisfies these tests, it can then be instantiated in the target problem. GTMs specify the patterned modifications not in terms of the physical structure of devices but in terms of their causal behaviors. Therefore, a GTM is required to be instantiated in the causal behavior of the source design. The MBA process modifies the structure of the design only after verifying that the modification satisfies the target problem constraints by simulating the modified behavior(s). Thus, it not only generates a solution to the new problem but also generates a functional and causal explanation (in the form of an SBF model) for how the solution satisfies the requirements of the problem.

2.3.3 Evaluation of the Solution

We refer to a design solution proposed for the target problem as a *candidate design* while we refer to a solution for the target as a *target or desired design*. Since the modifications proposed to a source design may not always result in a design solution, it is best to first verify them before making them to the structure itself. The main issue is how to do the evaluation. One method for evaluating a candidate design is by qualitative simulation. The knowledge of the SBF model of the design enables this method in MBA. We call this method internal evaluation. Clearly, this method is limited by the knowledge the MBA process has. Another method for evaluating a candidate design is by implementing the design in the real world and verifying how the design meets the given constraints. This method requires interaction with an oracle to find the outcome of implementing the design. We call the second method external evaluation.

2.3.3.1 Internal Evaluation and Design Failures

Since in our theory, the structure of a design is modified only after verifying that the proposed modifications work, the internal evaluation of the design is interleaved with the transfer and modification. Therefore, once the behavior of the candidate design is generated, MBA evaluates the design by qualitative simulation of the SBF model of the design. Qualitative simulation, as in the KRITIK system, involves tracing through the states and transitions in the model from the modified state to the final state by substituting new values for the parameters and checking if the desired function is achieved. If MBA finds that the candidate design (i.e., the proposed solution) delivers the desired functions, it goes to the next step of learning from the source and the target designs.

Suppose the process finds that the design does not satisfy the constraints of the given problem. Then the question is where should the control go in the MBA process. There are several paths for the flow of control in the process. In IDEAL, we developed a particular control flow—that involves backtracking to the nearest previous decision point in the process in case of failure. Figure 2.1 shows the backward-flowing control paths from the evaluation task to the spawning of adaptation goals and to the retrieval of a source analogue. That is, when the qualitative simulation identifies that the candidate design fails, the MBA process in IDEAL first tries an alternative adaptation strategy if one is available for the same adaptation goal. An example would be applying the cascading GTM to the same substructure in a source design after trying component replacement. If no alternative strategies are applicable, then it tries to make a different modification to the source design if there are alternative modifications identified that could result in reducing the functional differences. An example would be modifying bulb instead of battery in a simple electric circuit to deliver a different intensity of light. If no alternative modifications are possible, then it tries to adapt a different source design. An example now would be given the problem of designing a high-acidity Sulfuric Acid cooler, making an analogy from the design of a high-acidity Nitric Acid cooler after having tried unsuccessfully from the design

of a low-acidity Sulfuric Acid cooler. When there are no alternative source designs available for the given problem or they do not lead to a target solution, the process fails.

2.3.3.2 External Evaluation and Design Failures

Since the MBA process uses the SBF model of the candidate design to verify if the design works and the SBF model may be incomplete, it may not be possible to detect some design failures by simulating the model (in internal evaluation). Also, some design failures may not be detected internally for various reasons: the initial problem specifications may not always be complete; they may not clearly indicate the constraints from the environments in which the designs are intended to work; and the intended environments may themselves have changed from the time of problem specification to the time of design use. However, if an oracle interacts with the process and presents feedback on failures of the design (i.e., the outcome of implementing the design), then the feedback can be interpreted and the solution redesigned.

Knowledge Acquisition and Problem Reformulation: When the MBA process interacts with an oracle and acquires feedback on the design failures, then its first step is to interpret and understand the feedback. In MBA, the design failures are represented as failure behaviors, i.e., undesired behavioral state transitions, in the SBF language. One method for understanding the design failures is to form causal explanations for them. Then the question is what knowledge might the MBA process use for this purpose. We explored how GPPs, another interesting type of design patterns besides GTMs, can be retrieved and instantiated for this purpose because of the causal relationships that the GPPs capture. As in the indexing of GTMs, the task for which the GPPs are used determines the indexing scheme for the GPPs. In MBA, GPPs are indexed by their behavioral abstractions and are accessed by design failures as probes into the memory of GPPs. The causal explanations formed by instantiating GPPs enable the MBA process to reformulate the design problems if necessary. One way problems can be reformulated is by the discovery of new constraints and addition of those constraints to the problems. Problem reformulation may also involve deletion and modification of constraints in the problems.

Once the target problem is reformulated, there are two possible approaches to complete the design: (1) abandon the failed candidate design and look for a new source design to adapt or (2) redesign the failed candidate design to incorporate the new constraints. In IDEAL, we explored the second approach. Generating a design for the reformulated target problem requires designing for the new constraints and composing the new sub-designs with the failed candidate design. Figure 2.1 shows the control flow from the external feedback to redesign via the task of understanding design failures and the task of reformulating problems—depending on whether there is any reformulation of the problem, the redesign task may involve recursively applying the step of analogue retrieval (i.e., the retrieval of a source analogue) and the subsequent steps of analogy. Thus the MBA process can acquire the knowledge of design failures from an external

agent, reformulate the problems of the failed designs, and redesign the failed designs.

2.3.4 Learning from the Source and Target Analogues

The MBA process relies on different types of knowledge. One issue is where does this knowledge come from. In MBA, different kinds of knowledge may be learned from the experience of solving a target problem—whether it be success or failure. The knowledge that can be learned may include the design solution itself, a model for the design solution, and an abstraction over the models of the source and the target design analogues (such as a design pattern). Learning in both successful and failed problem-solving situations is justified and useful—for instance, learning some design patterns under failure can enable the process to avoid similar failures in future; and learning some design patterns under both success and failure can enable the process to do cross-domain transfer in future.

2.3.4.1 Problem-Solving Failures and Knowledge Acquisition

When the MBA process evaluates a candidate design by simulating the SBF model of the design and finds that the design does not satisfy the constraints of the problem, it tries different alternatives for fixing the design (as described above in Section 2.3.3.1). If no alternatives result in solving the problem, the process fails. Then there is an opportunity for the MBA process to learn knowledge that can help the process to avoid similar failures in future. But the issue is where does it acquire knowledge that enables its learning. One method is interaction with an oracle. That is, at such a problem-solving failure, the process can interact with an oracle (i.e., an external agent) in different ways to acquire knowledge and learn from the failure. We explored different interaction conditions where the information presented by the oracle varies. The oracle can present the MBA process different kinds of information on a solution to the target problem: (1) the complete structure of the target design that satisfies the problem constraints and an SBF model for the design, (2) only the complete structure of the target design, or (3) only the localized structure for the target design (i.e., a solution to the specific, local adaptation goal). Under all these conditions, the MBA process can acquire a solution to the target problem and learn a new GTM from the source and the target design analogues. Because GTMs encapsulate strategic knowledge, the problem-solving failures (more precisely, the failures in adaptation) provide a good context for learning the GTMs.

Comparing the source and target design analogues and abstracting the relevant functional and causal relationships to form a useful design pattern can be very complex. The complexity may be partly due to the dimensions along which the designs can differ and partly due to the complexity of the designs. The MBA process uses the SBF models of the source and target designs to keep the learning of design patterns computationally tractable. Since the learning of design patterns in MBA is thus model-based and in some of the above interaction conditions the SBF model for the new design is not available, there is a need for learning the SBF model

itself. In MBA, different methods are used for acquiring the SBF models of new designs. When the MBA process tries to modify a source design, it first generates the SBF model for the target design by revising the model of the source design. MBA has two different methods for that: (1) simple revision of the model of the source design (Goel, 1991b) and (2) complex revision of the model by the instantiation of design patterns in the model of the source design. These two methods are feasible when the modification of the source design is successful, i.e., it results in a design for the target problem. But when the MBA process fails to generate a design solution, it requires different methods to learn an SBF model of the target design. The MBA process interacts with an oracle and acquires the different kinds of information listed above. When the oracle directly gives the SBF model for the new design, the acquisition is trivial. But in the other two conditions, the MBA process uses a new method for learning SBF models. The new method we developed combines the methods of model revision and composition of behaviors of primitive structural elements.

2.3.4.2 Model-Based Learning of Design Patterns

The task of learning design patterns takes as input a target design analogue and a source design analogue and gives as output a design pattern such as GTM and GPP. When the target design that solves the given problem is available (whether generated automatically or given by an oracle), the MBA process can abstract over the source and the target analogues to acquire a new design pattern. The issue is what knowledge in the source and target enables the learning of design patterns. Because of the knowledge (i.e., relationships) captured in design patterns, the MBA process uses the SBF models of the source and target designs to learn the design patterns. It compares the SBF models of the two analogues along all dimensions of the representation and discovers patterns of regularity in the functions and behaviors of the two analogues. The MBA process does not have access to any a priori knowledge of the patterns it learns. Therefore, its goal is to identify what differences in the behaviors of the two designs might be responsible for the differences in the functions of the designs. Such a "blame-assignment" task can be quite complex depending on the complexity of the designs. Therefore, this comparison in MBA is constrained by the internal organization of the SBF models and the problem-solving context in which the learning occurs. For instance, if the MBA process failed in the context of modifying a substructure of the source design and received the target design from an oracle, then the behavior segments in the SBF models that correspond to the source and target substructures are relevant for comparison, and the process focuses on them. Once it finds some regularity between the designs, the MBA process abstracts over them along the dimension of structure and forms the relationships between functions and behaviors as design patterns. The reason for abstracting along the dimension of structure is that the structure characterizes and represents the specificity of a domain.

Although in Figure 2.1 learning is shown as separate from the problem-solving steps (i.e.,

transfer and modification, and evaluation of the solution), much of the work considered part of learning—for example, generating the explanation of how a solution satisfies the constraints of the given problem (i.e., the SBF model)—is actually done during problem solving, and thus the separation is only functional.

Learning at Different Stages of Analogy: In Figure 2.1, learning is shown to occur at the time of storing a target design analogue. That is merely an artifact of the current implementation of the MBA process in the system IDEAL, but the theory makes no commitment to such an architecture. Learning could very well occur at the time of problem solving using the same model-based method, although it involves a different scheme of indexing design analogues and accessing them in the context of a target problem. When a source analogue is distant from the target problem, transferring knowledge from the source involves abstracting relevant information in the model of the source analogue and instantiating the abstract model in the target problem. The relevant information for abstraction would be the relations in the model of the source analogue that were functionally responsible for solving the source problem. The rationale is that the information that is functionally important in the source domain when abstracted and instantiated into the target domain would solve the target problem because the source problem and the target problem are similar. The same rationale underlies the above learning method. Abstractions necessitated by problem solving and formed during problem solving can also be stored in memory for later use.

Yet another stage for learning abstractions is the retrieval of a source analogue itself! Although it may seem possible and useful in certain tasks such as explanation completion, it does not appear to lead to abstractions of the kind this research addresses in tasks such as design. If any, only the abstractions of target problems seem possible during retrieval because there would be no target analogues available at that stage! But, in general, abstractions can thus be learned at different stages in the process of analogical reasoning: during the retrieval of a source analogue, during the transfer and modification of a source analogue, and during the storage of a target analogue. In this research, we focused on learning of abstractions such as design patterns while storing target analogues in memory.

When a design pattern is formed from the source and target design analogues, it is only postulated as a hypothesis which can be revised based on later experiences of using it. If the design pattern is used to generate a solution in a later problem-solving situation and the solution is evaluated to have failed, the design pattern may be revised based on feedback from an oracle. When the oracle presents the correct solution to the problem, following the same learning process described above, a new hypothesis for the design pattern could be postulated. The new design pattern can then be assimilated (i.e., merged or kept separate) with the old design pattern which may result in generalizing the old pattern further or refining it.

2.3.5 Storage of the Target Analogue and the Design Pattern

Finally, the MBA process involves storing the target design analogue and the learned design pattern (if any) into its memory for potential reuse. In our theory, memory is an important component of the process of analogy. The organization and indexing of design analogues and design patterns in memory can influence whether some subprocesses of MBA occur or they do not occur. Since the utility of a piece of knowledge learned depends on learning the right *indices* for the knowledge so that it can be brought to bear at the right time, this subtask involves index learning too. Thus, the MBA process first identifies the appropriate indices for the target design analogue and the design pattern. In MBA, the SBF models together with the task context suggest what features should be used as indices for design analogues and design patterns, and enable automatic, dynamic acquisition of those indices. Since the MBA process has only a few design patterns, it currently does not organize them hierarchically (because the efficiency of their retrieval is not an issue). However, in order to enable an efficient and effective retrieval, it organizes design analogues in multiple hierarchies where each hierarchy is along a feature in the function or structure of the design analogues.

2.4 Model-Based Analogical Design: An Illustrative Example

We will now illustrate the process of MBA for design with an example of learning and use of a specific design pattern, i.e., one type of feedback GTM, from IDEAL. The specific examples of design problems and the GTM are from the computer program IDEAL. That is, the MBA process in this description refers to the process in IDEAL. Figure 2.2 presents the complete story of this illustration. The story has two parts. As we present the two parts, one describing how the feedback GTM is learned in the domain of electronic circuits and the other describing how the learned feedback GTM is used as an adaptation strategy in designing in the domain of mechanical controllers, we walk through the different steps of the process (that are relevant to the example) twice. The story goes through the following sequence. First, given a target problem in the domain of electronic circuits, the MBA process retrieves a past, similar design that is a partial match to the problem. It then identifies the differences between the functions in the target problem and the source design and tries to form the adaptation goal. The target design problem and the source design are such that the MBA process cannot localize the functional difference to making a modification to a component. Hence, its attempts to perform simple modifications such as component replacement or substance substitution fail. Suppose that the MBA process does not have the knowledge of any GTM that matches the functional difference to be reduced. (The particular GTM needed is the feedback GTM and the process does not have it at this time.) The process fails because it cannot make alternative modifications to the source design. Then the process can interact with an oracle to receive the design solution and an SBF

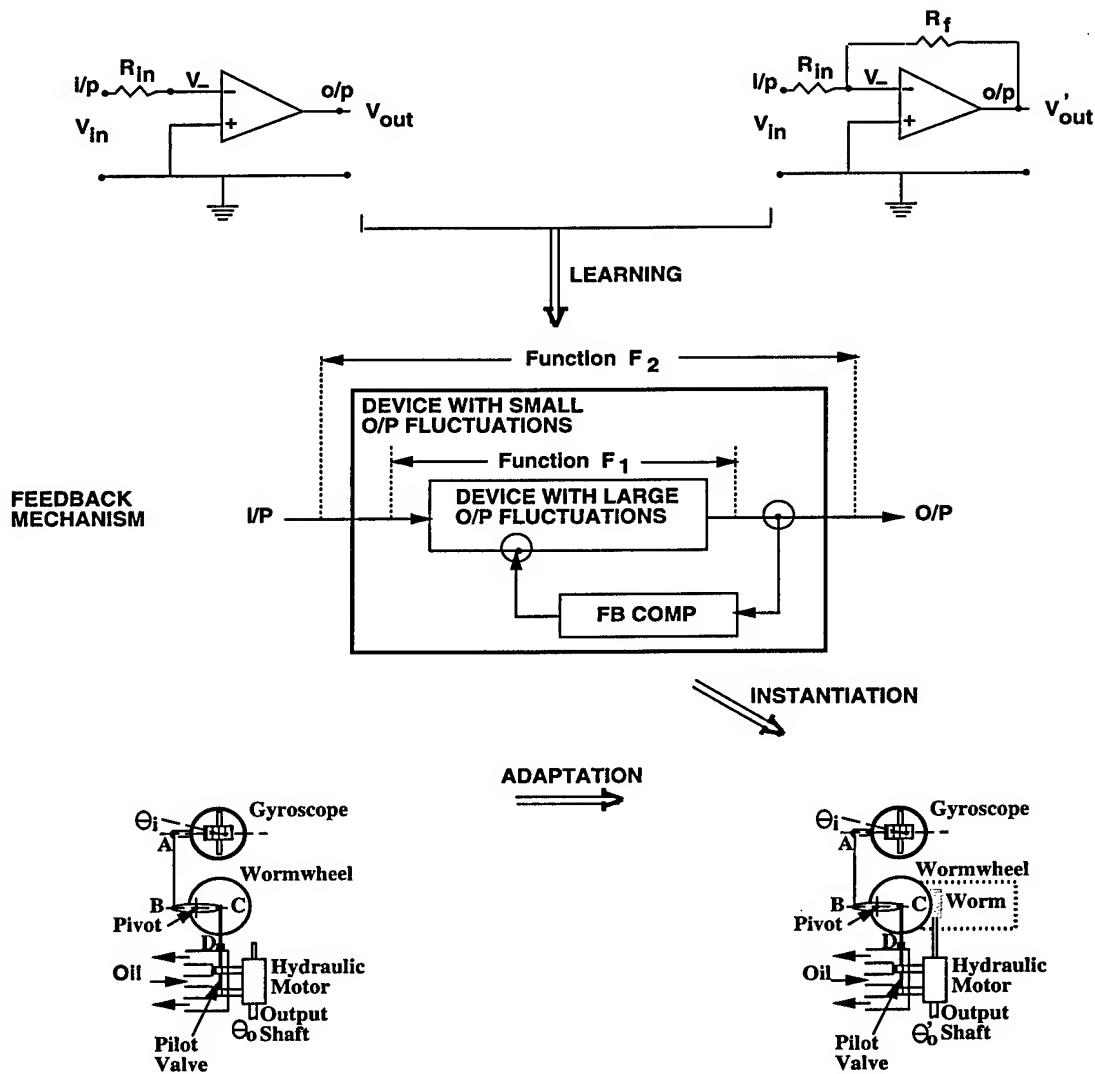


Figure 2.2: An Illustration of the Process of Model-Based Analogical Design via Learning and Use of A GTM that We Call Feedback

model of the design.¹ The MBA process then learns a GTM (that we call feedback GTM) from the source design and the target design by abstracting over a regularity in their SBF models so that the GTM is useful to avoid similar failures in future. The process also learns an abstraction over the current functional difference as an index to the learned GTM and stores the GTM in its memory.

In the second walk-through of the process, the MBA solves a problem in the design domain of mechanical controllers. This time, however, the process is able to access the learned feedback GTM when it tries to solve the adaptation goal (which is not local) and is able to apply the strategy of the GTM successfully. It instantiates the GTM in the context of the SBF model of the design of a mechanical controller (i.e., the source design) and generates first an SBF model of the target design. It simulates the SBF model to verify if the proposed design achieves the desired function and then modifies the structure of the design. Since there is no problem-solving failure at this time, the MBA process does not interact with the oracle and does not learn a new GTM. However, it identifies indices for the target design and stores the design in the memory of analogues.

Let us now consider the first walk-through of the process of analogical design. Suppose that the MBA process is given the problem of designing an electronic circuit whose function is to produce an electricity with a voltage value, V'_{out} ($= V_{avg} \pm \delta$, where δ represents a small fluctuation over an average value V_{avg}), taking an electricity of voltage V_{in} as input. Figure 2.3 shows this desired function in the SBF language.

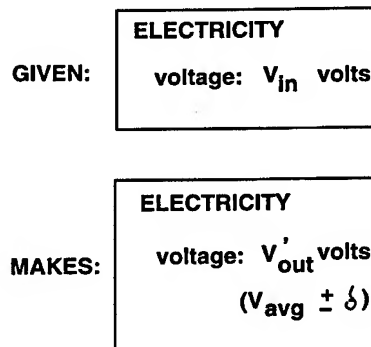


Figure 2.3: Desired Function of A New Electronic Circuit

Given the above problem, the MBA process first probes its memory of design analogues to retrieve matching designs. Suppose that its memory contains the design of a simple amplifier whose structure is shown schematically in Figure 2.2(top-left). Figures 2.4 & 2.5 respectively illustrate the function and the behavior of the simple amplifier. The function of the design in

¹We are only considering one interaction condition for the simplicity of the illustration.

memory specifies that the device takes as input an electricity of voltage V_{in} volts at the location i/p and gives as output an electricity of voltage V_{out} volts (i.e., $V_{avg} \pm \Delta$ where Δ is a large fluctuation around an average value and $V_{out} > V_{in}$) at the o/p location. The function also points to the internal causal behavior of the device shown as a sequence of states and state transitions in Figure 2.5. The behavior explains how the structure of the device achieves the

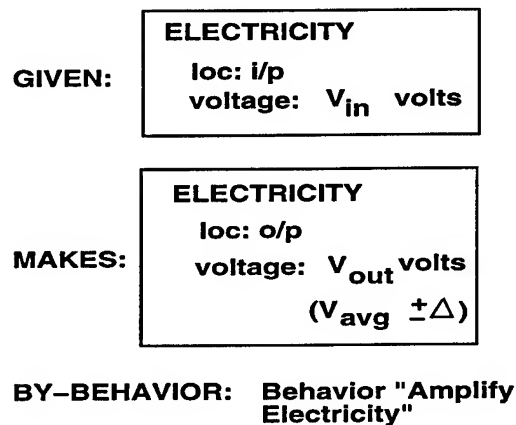


Figure 2.4: Function "Amplify Electricity" of A Simple Amplifier

function, i.e., how the input electricity is amplified at the output. Note that each transition is annotated with the function of a structural component that contributes to that transition and with the

qualitative relationships between state variables.² MBA retrieves this design of a simple amplifier when it probes its memory with the desired function because the functions match on the property "voltage" specified in them. However, the match is only partial because the MAKES states in the two functions do not match on the voltage value.

Since there are differences in the functions of the desired design and the source design, the MBA process now tries to form an adaptation goal. The functional difference is in the range of fluctuation of an output property value (i.e., while the source design's output fluctuates over a large range, the desired output fluctuation is small). The fluctuations in the output of a device can, in general, arise due to several reasons, for instance, due to the fluctuations in the input of the device or due to unstable device parameters. The MBA process, using the SBF model of the source design and the functional difference, tries to identify which structural components in the source design may be modified in order to reduce the overall functional difference. But,

²In general, there can be many other types of information that may be specified in a transition and the SBF language provides primitives for that. We will see some of them in other examples in later chapters.

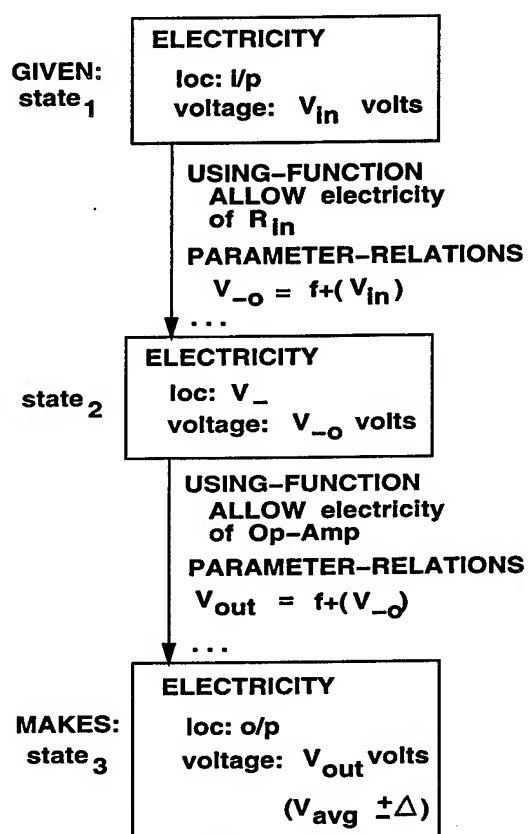


Figure 2.5: Behavior “Amplify Electricity” of A Simple Amplifier

in this particular example, it cannot localize the needed modification to any single structural component because the SBF model of the source design does not indicate any relationships between the fluctuations and the component parameters.³

Suppose that the MBA process only has simple strategies, such as replacing a component in a past design or substituting a new substance for one in a past design, to deliver new functions. In the current scenario, none of these simple strategies is applicable. That is, given the adaptation goal of reducing the specific functional difference, which is *large* vs. *small* fluctuation in the output property value, the MBA process cannot find any of the adaptation strategies to be applicable. Since there are no alternatives available for modification, including a different source design, the process fails.

Next, the MBA process interacts with an oracle to receive the design solution for the target problem and an SBF model for the design. Figure 2.2(top-right) shows the structure of the target design schematically and Figure 2.6 shows the internal behavior of the design.

Given the SBF models of the source design (i.e., the simple amplifier) and the target design, the MBA process compares them state-by-state and transition-by-transition along all possible dimensions in their SBF representation in order to identify the regularity between them. That is, it performs a differential diagnosis on the two models by which it determines (1) if there is a behavior segment in the SBF model of the target design that matches with the SBF model of the source, and (2) if so, what additional segments there are in the SBF model of the target and how they are related to the matching segment. Once the MBA process determines those relationships, it abstracts over the specific substances, properties and values in the relationships and forms a GTM that encapsulates the abstracted functional differences and the abstracted (causal) behavioral relationships. The mapping between the functional differences and the behavioral relationships in the new GTM will help the MBA process avoid failures similar to the current one. That is, they would suggest how to modify the behavior of a source (candidate) design in order to generate the behavior of a target (desired) design, and reduce the functional difference between them.

Figure 2.7 illustrates the SBF representation of the new GTM that the MBA process learns. Note that this representation does not refer to any specific substances or components, or their properties. But the functional and causal relationships between the specific source design and the target design are preserved: the left half of the figure illustrates the functional differences and the right half the behavioral relationships. The MBA process indexes the new GTM by the functional differences the GTM reduces, and stores the GTM in a flat memory.

Finally, in order to store the target design in analogue memory for later use, the MBA process first identifies the appropriate indices for the design. Using the knowledge in the SBF model of the target design, it selects only those features in the function of the design that are relevant to

³Even if there is a relationship between the open-loop gain of the op-amp and the fluctuation, and if the op-amp can be selected as a localized component to modify, replacing the op-amp with another will not satisfy the constraint that the output fluctuation be small.

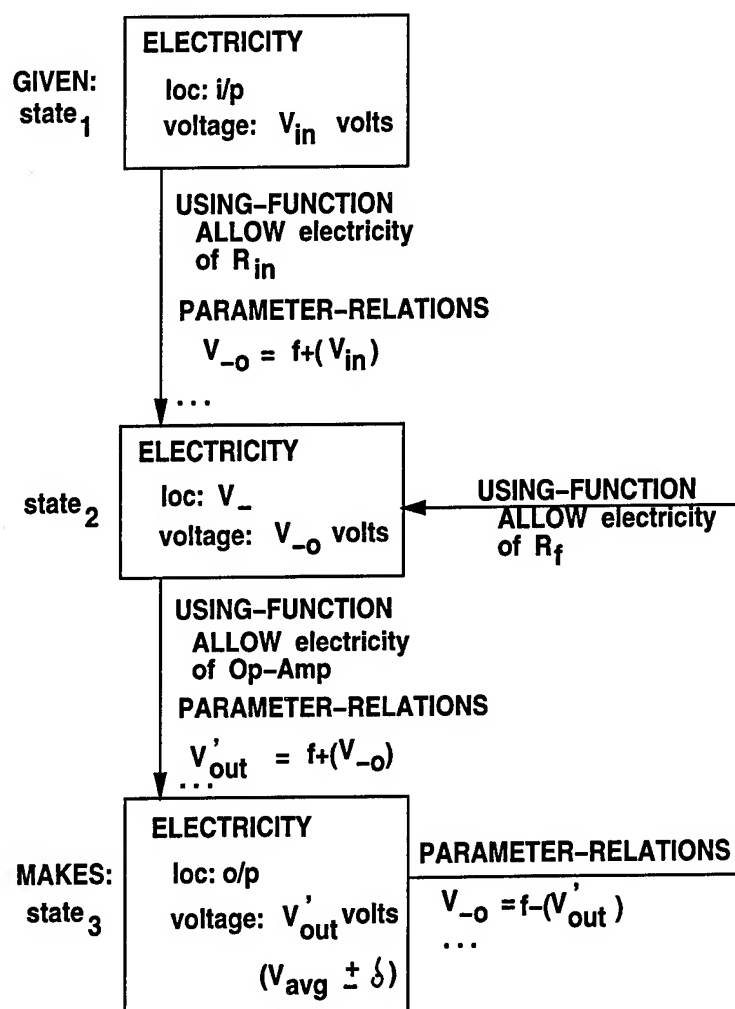


Figure 2.6: Behavior "Amplify Electricity" of the Desired Design

the working of the device. In the current example, however, the process stores the target design along the dimension *voltage* which is the only property specified in the function.⁴

That completes our first walk-through of the MBA process. Let us now consider the second walk-through of the process. Suppose now that the MBA process is given the problem of designing a gyroscope control system whose function is to produce an angular momentum with a magnitude, $L'_o (= L_{avg} \pm \delta)$, where δ represents a small fluctuation over an average value L_{avg} , taking an input angular momentum of magnitude L_{in} . Figure 2.8 shows this desired function in the SBF language.

Given the above problem, the MBA process first probes its memory of design analogues to retrieve matching designs. Suppose that its memory contains the design of a simple gyroscope

⁴We will describe a more interesting index learning situation from the MBA process in a later chapter on memory.

DESIRED DESIGN:

GIVEN: ?SUB
?prop1: ?val11

 F_2

MAKES: ?SUB
?prop1: ?val22

BY-BEHAVIOR: Behavior B2

CANDIDATE DESIGN:

GIVEN: ?SUB
?prop1: ?val11

 F_1

MAKES: ?SUB
?prop1: ?val21

BY-BEHAVIOR: Behavior B1

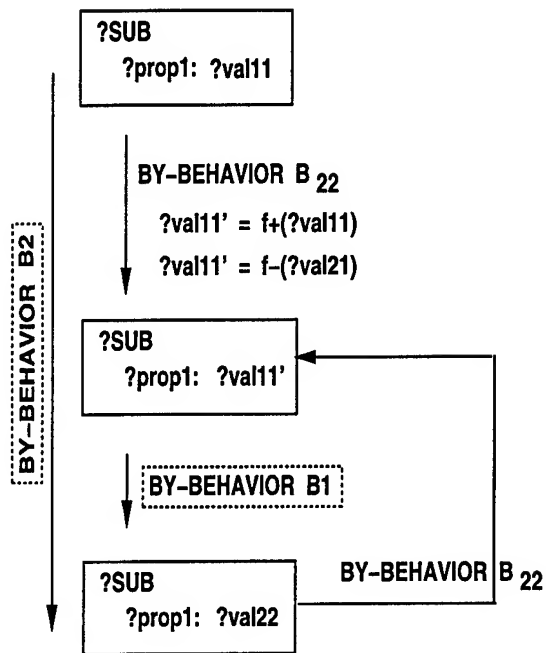
CONDITION:

$$?val22 \neq ?val21 \rightarrow ?val21 = ?val \pm \Delta$$

$$?val22 = ?val \pm \delta$$

$$F_2 = f : (?val11, ?val21) \rightarrow ?val11'$$

$$+ F_1 : (?val11) \rightarrow ?val21$$



$$B2 = B1 + B22$$

where B22 achieves function f

The relationships between B1 and B22 are such that:

$$\text{FINAL-STATE (B1)} \leftarrow \text{INITIAL-STATES (B22)}$$

$$\text{FINAL-STATE (B22)} \leftarrow \text{STATES (B1)}$$

Figure 2.7: SBF Model of the Feedback GTM Learned from the Designs of the Simple Amplifier & the New Device

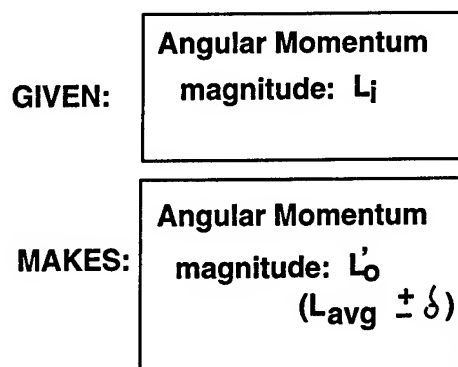


Figure 2.8: Desired Function of the New Gyroscope Control System

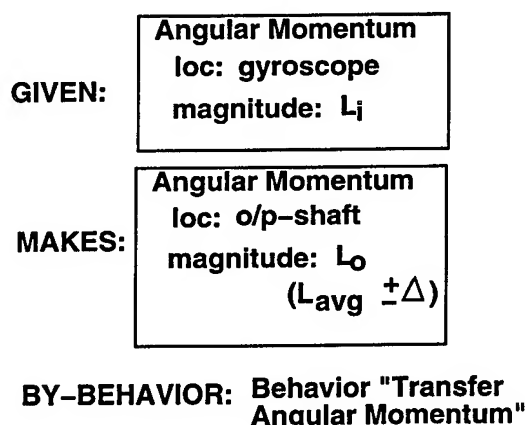


Figure 2.9: Function "Transfer Angular Momentum" of A Design Analogue (a simple Gyroscope Control System)

control system whose structure is shown schematically in Figure 2.2(bottom-left). Figures 2.9 & 2.10 respectively illustrate the function and behavior of the simple gyroscope control system. The function of the available design is similar to the desired function except that the output angular momentum fluctuates over a larger range. The function also points to the internal causal behavior of the device that explains how the structure of the device achieves the function. The MBA process retrieves this design of a simple gyroscope control system because the functions of the target problem and the source design are similar. But, of course, the match is only partial as it was in the previous scenario because the MAKES states in the two functions do not match on the magnitude of angular momentum.

Like in the previous scenario, since there are differences in the functions of the target design and the source design, the MBA process now tries to form an adaptation goal. Again, using the SBF model of the source design, the process tries to localize the required modification but in vain. It first checks if any of the simple adaptation strategies applies in the current context

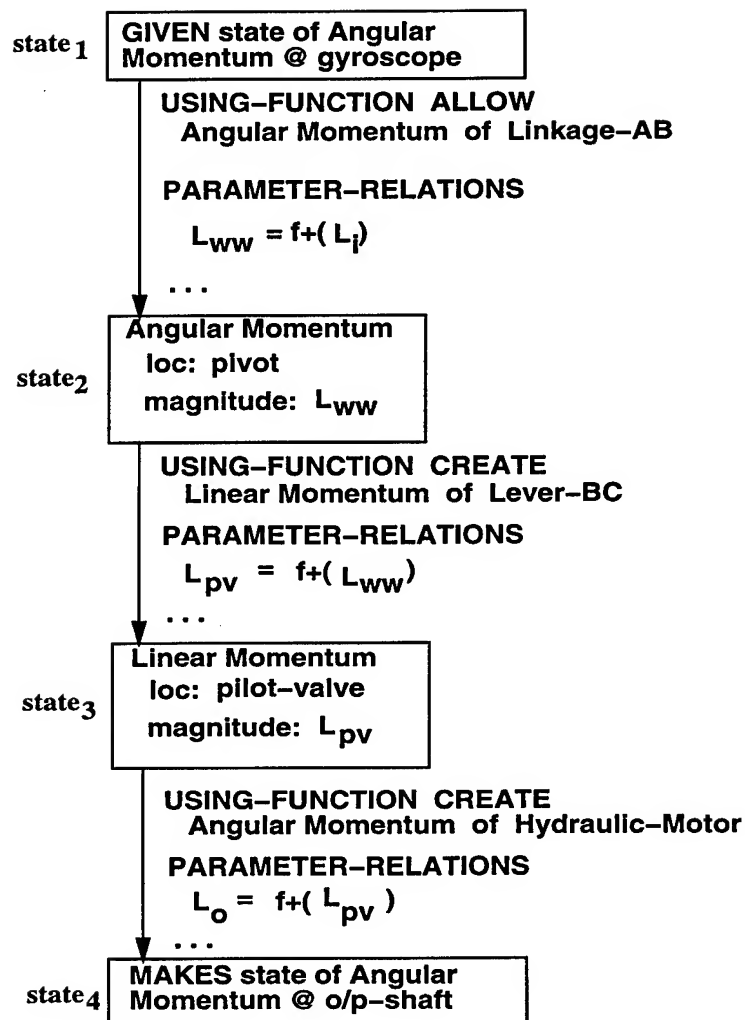


Figure 2.10: Behavior "Transfer Angular Momentum" of the Design Analogue (a simple Gyroscope Control System)

and finds that none does. Unlike before, however, since the MBA process now knows a GTM, a more complex design adaptation strategy, the process tries to probe the memory with the current adaptation goal. It now finds that the adaptation goal (characterized in terms of the overall functional difference to be reduced) matches with the index of the feedback GTM it had learned from the designs of electronic circuits. The match is successful because the index of the GTM is specified in abstract, generic terms and the type of functional difference in both the adaptation goal and the GTM are the same (i.e., *large* vs. *small* fluctuation in the output property value).

Then, in order to instantiate the GTM, the MBA process needs to match the decomposability condition on the desired function in the GTM (see Figure 2.7 for the condition $F_2 = \dots$) with the current desired function and find the subfunction f that needs to be designed for and composed with the function of the source design. From the initial match of the functions, the MBA process gets bindings for the two input states of the subfunction. But, in order to find the bindings for the output state of the subfunction, the process needs to backtrace the SBF model of the source design and find an intermediate state that can match the output state of the subfunction. In the current example, there are only two possible candidate states, $state_2$ and $state_3$ in Figure 2.10 for this purpose. The choice between the two is very simple because $state_3$ describes a different substance, namely, linear momentum, rather than angular momentum. Hence, the MBA process selects $state_2$.⁵ Now, instantiating the "template" subfunction from the GTM with the bindings from $state_1$, $state_4$, and $state_2$ of the behavior of the source design, the process formulates the specific subfunction shown in Figure 2.11. Note that the relationships in the subfunction come from the knowledge in the GTM. Informally, the subfunction is to produce an angular momentum with a different magnitude (L'_{ww}) at the location of pivot, given the angular momentum (L_i) at the gyroscope location and the angular momentum at the o/p-shaft location that fluctuates over a large range ($L_o = L_{avg} \pm \Delta$).

To complete the instantiation of the GTM in the context of the source design, the MBA process needs to solve the subproblem, i.e., design for the subfunction, and compose the behavior of the subdesign with the behavior of the simple Gyroscope Control System as per the relationships specified in the GTM. In the current design scenario, the subfunction for which the process needs to design really has two parts (because the subfunction specifies two inputs and one output). But the behavior of the source design already explains how one of those state transformations ($state_1 \rightarrow state_2$) is achieved, and hence the MBA process only needs to design for the other state transformation (i.e., the transformation from the state of angular momentum with magnitude L_o to the state of angular momentum with magnitude L'_{ww}). Suppose now that the process has the knowledge of a component (called worm) whose function exactly matches the desired part of the subfunction. After substituting the appropriate parameters in the behavior of the retrieved subdesign (i.e., worm), the process composes it with the behavior of the

⁵But, in general, if there are multiple states all of which refer to the same substance, then the heuristic for selection is that the state causally nearest to the MAKES state should be chosen.

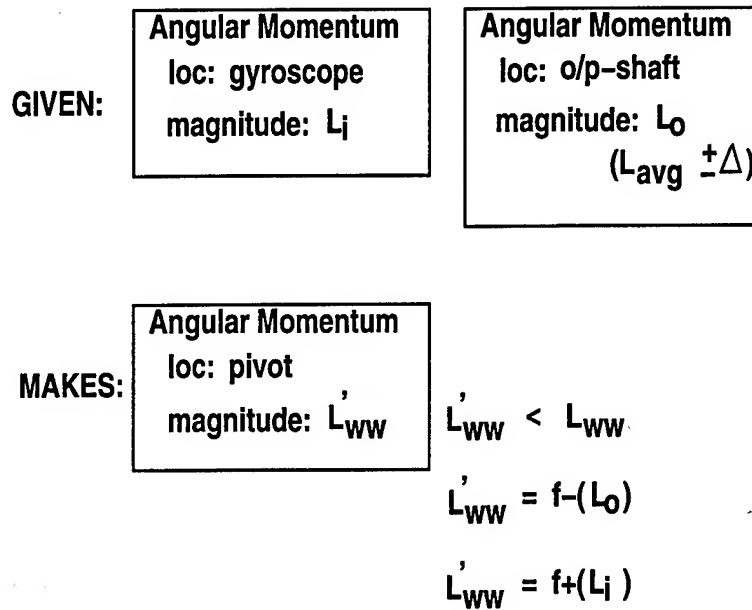


Figure 2.11: The subfunction formed by the Instantiation of the Feedback GTM

initial source design (i.e., simple Gyroscope Control System) as per the behavioral relationships specified in the GTM and produces the behavior of the target design shown in Figure 2.12.

Next, the MBA process propagates the changes in the values of the state variables due to the behavior composition forward in the causal behavior (until the end of the behavior is reached or until the same state repeats) and verifies if the modified behavior achieves the desired function. Note that it also propagates the changes to the other behaviors in the model of the device that are dependent on the currently modified behavior. In the current example, the new value for the magnitude of angular momentum in *state*₂ (i.e., L'_{ww}) is propagated forward through the states *state*₃ and *state*₄. Once the MBA process thus simulates the behavior, it modifies the structure of the source design. The structure of the target design is shown schematically in Figure 2.2(bottom-right).

Since the MBA process is now able to successfully use the GTM in solving the current design problem, it does not learn any new abstraction. The process finally identifies the indices for the target design analogue and stores it in memory along the dimension *magnitude* of angular momentum. That of course completes our second walk-through of the MBA process for design. As illustrated in this two-part story, the MBA process can learn a GTM in one domain such as electronic circuits and use it in another such as mechanical controllers. Thus, the MBA process can do an interesting cross-domain transfer via GTMs. Note however that the SBF models of devices also play a significant role in different steps of the MBA process for design.

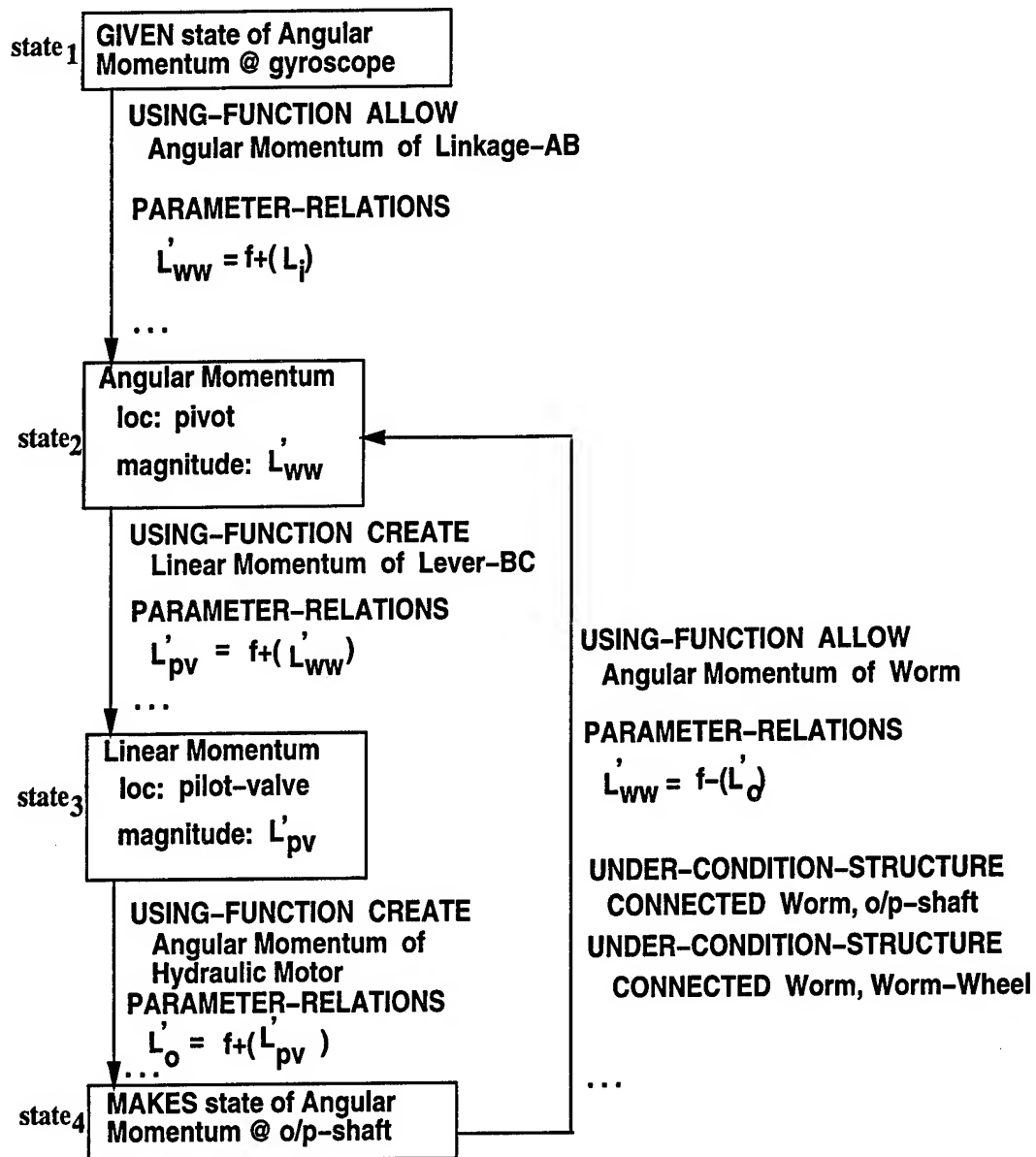


Figure 2.12: Behavior of the New Design Achieved by Composing the Behaviors of the Design Analogue (i.e., simple Gyroscope Control System) and the Subdesign (i.e., worm)

CHAPTER III

THE CONTENT OF DEVICE MODELS AND DESIGN PATTERNS

As we described in the previous two chapters, two different types of models play significant roles in different stages of analogical design: SBF models of specific devices and of design patterns. In addition, our theory of analogical design uses knowledge of primitive components, substances, and functions in design domains. In order to enable the three facets of innovative design, i.e., non-local modifications, cross-domain transfer, and problem reformulation, and address the complexity in reasoning required, the MBA process requires the knowledge of design patterns. Some useful design patterns in the context of device design are high-level design abstractions that capture relationships between functions and behaviors of devices (in particular, patterns of those relations). Therefore, to represent design patterns, we need primitives that capture functions and behaviors of devices and regularities in their relationships. We adopt the Structure-Behavior-Function (SBF) representations (Goel, 1989) for this purpose.

In order to explore our hypothesis about the learning of design patterns, we need to represent device models too. A device model encapsulates functions and behaviors of the device in terms of its specific structure. The theory of SBF models was developed precisely to capture device models. We also represent device models in the form of SBF models. The SBF model of a device if available in a design analogue not only enables learning of design patterns, but also enables other subtasks of analogy such as adaptation (Goel, 1991a) and evaluation of a solution (Goel, 1991b).

In this chapter, we will first describe the content of device models in terms of SBF representations. The SBF representation in IDEAL is based on the previous work on KRITIK (Goel, 1989) and KRITIK2 (Bhatta and Goel, 1992; Stroulia et al., 1992). Using the representation in this work has led us to add some new primitives to the original formulation of SBF (representation) language. We will indicate them in this chapter where appropriate. Then we will describe how the SBF language can be used and extended for representing the content of design patterns, which was our focus in this research with respect to knowledge representation. In addition, we describe other types of conceptual knowledge, i.e., primitive functions, substances, and components, which form the basis for the SBF representations of device models.

3.1 Structure-Behavior-Function Models of Devices

SBF models of devices specify how the causal relationships between the specific structural elements in the devices result in the devices' functions (i.e., output behaviors). The primary characteristic of these models is that they capture *teleological*, *causal*, and *structural* knowledge of devices. The SBF model of a device captures the designer's comprehension of how the device works, that is, how the functions of structural elements get composed into the functions of the overall structure. These models are based on a *component-substance ontology* (Bylander and Chandrasekaran, 1985). This ontology gives rise to the SBF language (Goel, 1989, 1991a) for describing the model of a design that is a generalization on Sembugamoorthy and Chandrasekaran's (1986) functional representation scheme. The constituents of the SBF model—structure of the device (i.e., the physical structure), the functions delivered by the structure, and the internal causal behaviors—are described below.

In addition to the schema-like descriptions of the constituents of SBF models, we also present a formal specification of them.¹ In the formal notation, an n -tuple representation for a type of knowledge indicates that this type of knowledge has n constituents. For example, a case (or an analogue) is a 3-tuple where the elements are function F , structure S , and an SBF model M ; it is also implicit that these three constituents for a case are "coherent" in that the model M specifies the internal causal behaviors that explain how the structure S delivers the function F . Each tuple here corresponds to a schema in the IDEAL system and each element in the tuple corresponds to a slot in the schema.

$$Case = (F, S, M)$$

Each of the constituents is recursively described/defined in terms of its lower-level constituents. In the following descriptions, all the enumerated entities are intended to be only partial sets that were necessary to deal with the class of devices IDEAL represents. In general, it is an empirical question as to how many different classes of devices these partial sets can cover. The constituents of the SBF models and their formal specification are now given below.

3.1.1 Structure

The structure of a design is expressed in terms of its constituent components and substances and interactions between them. Figure 3.1 shows a Sulfuric Acid Cooler (SAC) and Figure 3.2 its structure schema. Components and substances can interact both *structurally* and *behaviorally*. For example, in SAC, water can flow from H_2O -pipe to heat-exchange chamber only if they are *connected*, and Sulfuric Acid flows from $p1$ to $p2$ due to the behavior *allow* of H_2SO_4 -pipe-1.

Figure 3.3 shows the representation of schema for the device structure in the SBF language. The structure of a device is described hierarchically in terms of its constituent structural elements. The constituent elements of a device may be primitive domain components, such as a

¹The formalization was done in collaboration with Eleni Stroulia.

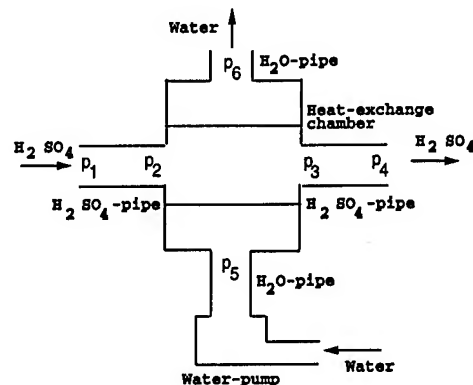


Figure 3.1: Sulfuric Acid Cooler

STRUCTURE SAC

COMPONENTS: (Heat-Exchange-Chamber
 H_2SO_4 pipe-1
 H_2SO_4 pipe-2
 Water-pump
 ...)

STRUCTURE Heat-Exchange-Chamber

RELATIONS: (SERIALLY-CONNECTED
 Heat-Exchange-Chamber
 H_2SO_4 pipe-1)
 (INCLUDES
 Heat-Exchange-Chamber
 H_2SO_4 pipe-2)
 ...

PARAMETERS: (volume v1)

FUNCTIONS: (ALLOW Water)

CONNECTING-POINTS: (p2 p3 p5 p6)

...

STRUCTURE H_2SO_4 pipe-2

RELATIONS: (SERIALLY-CONNECTED
 H_2SO_4 pipe-1
 H_2SO_4 pipe-2)
 ...

PARAMETERS: (Capacity R0)

FUNCTIONS: (ALLOW H_2SO_4)

CONNECTING-POINTS: (p2 p3)

...

Figure 3.2: The Structure Schema for the Sulfuric Acid Cooler

battery, or they may be complex structures, such as an air-conditioning unit, which can them-

selves be further described in terms of smaller constituent elements. Each structural element, except the overall structure of the device, points to another structural element of which it is a part. In addition to the *part-of* relation between structural elements, other structural relations, such as connectivity, inclusion, and containment are explicitly represented in the structure schema. The structural relations can currently be one of the four enumerated types: *contains*, *includes*, *serially-connected*, and *parallelly-connected*. The structural relationships, *contains* and *includes* are different in that the former specifies a relationship between a component and a substance while the latter specifies a relationship between two components. *Serially-connected* and *parallelly-connected* are also different: the former specifies a relationship between components such that the output of one becomes the input to the other, while the latter specifies that the two components share the same input and the same output. Figure 3.2 shows the structure of the Sulfuric Acid cooler and the specific components in it.

structure:

(components: A set of structural elements into which the structure under description can be decomposed.
 part-of: The larger device structure of which this is a part.
 structural-relations: A set of relations among the sub-elements of the structure under description.)

Figure 3.3: Structure Schema

Formally, a structure can be specified as

$$S = \text{Comp} || SA$$

which means that the structure can be a primitive component *Comp* OR (denoted by ||) an assembly of substructures *SA*.

An assembly of substructures itself can be specified formally and recursively as one or more of structures *S*, that is, $SA = \{S\}^+$. A primitive component *Comp* is a non-decomposable structure and is itself represented as a schema consisting of the slots: *is-a*, *modes*, *parameters* (component parameters and their values), *structural-relations*, *functions*, and *connecting-points*. A partial schema for the specific component *battery-1* is shown in the dashed box in Figure 3.12. *Is-a* links a specific component to the general knowledge of its prototype component in IDEAL's conceptual memory of components. Components are described in more detail in a later section.

Formally, a *Comp* is a 6-tuple whose elements are *is-a* (e.g., *H₂O-pipe is-a pipe*), zero or more *Modes* (e.g., *closed mode* of a Switch and *open mode* of a Valve), *Parameters* (as shown below), *structural-relations* (like those described above), *functions* (i.e., primitive functions delivered by the component), and *connecting-points* (the points on the component where other components in a structure can be connected). *Parameters* is one or more of triplets of component Parameter,

its Value and Units (e.g., the parameter *volume* of a heat-exchange chamber may have a value of 1 cu. ft.). IDEAL's knowledge about the primitive components in a specific device are linked to the general knowledge about the primitive components via the link *is-a*. Figure 3.12 shows IDEAL's partial memory of components.

Comp =

(*Is-a*, {*Mode*}*, *Parameters*, *Structural-Relations*, *Functions*, *Connecting-Points*)

Parameters = {(*Parameter*, *Value*, *Units*)}⁺

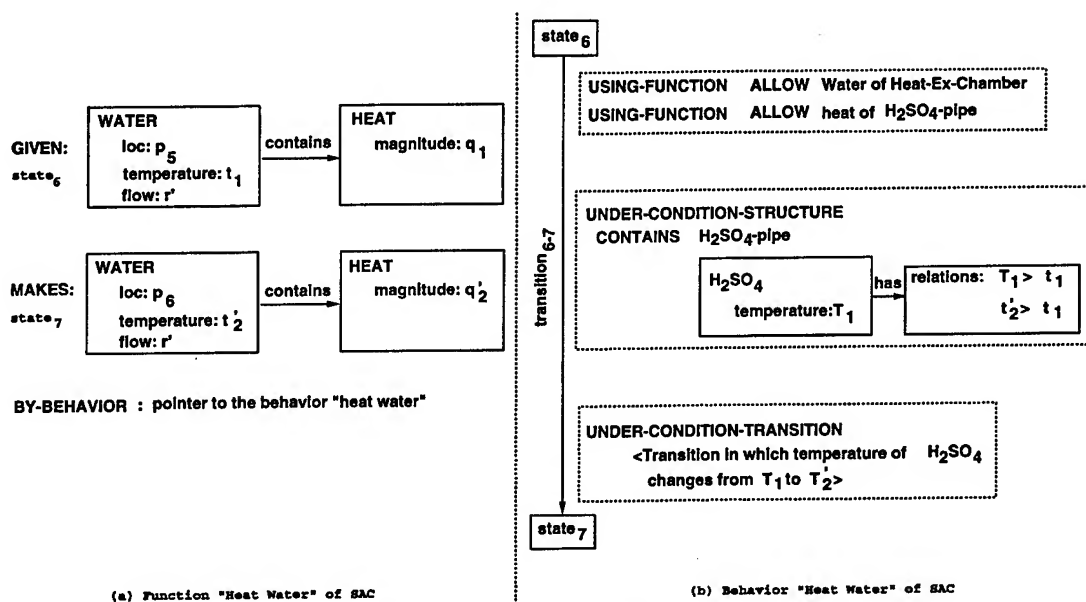
3.1.2 Function

A function is represented as a schema that specifies the behavioral state the function takes as input, the behavioral state it gives as output, and a pointer to the internal causal behavior of the design that achieves the function. Figure 3.4(a) shows a function of the SAC, namely, heating water. The input state of this primary function specifies that **water** at **location p5** in the topography of the device (Figure 3.1) has the properties **temperature** and **flow**, and corresponding values t_1 and r' . The input state also specifies that the **water** contains another substance **heat** whose magnitude is q_1 . Similarly, the output state specifies the properties and the corresponding values of the substance at **location p6**.

This representation of functions gives rise to a typology of functions in the domain: transformation functions, control functions, maintenance functions, and prevention functions. In this work, we primarily focus on transformation functions, which themselves are of several types such as substance transformation, substance-property transformation, and substance-location transformation. For example, the function of SAC is both a *substance-property transformation* and a *substance-location transformation* because it specifies a change in the value of the substance temperature as well as a change in the substance location. A substance-transformation function is one where the substance in the input state changes to a different substance in the output state. In addition to transformation functions, we also consider control functions, although implicitly, in the context of devices with feedback and feedforward mechanisms.

Figure 3.5 shows the schema for the representation of functions in IDEAL. Besides the input and output behavioral states (GIVEN and MAKES respectively), the schema for a functional specification contains a pointer to the internal causal behavior that transforms the input state into the output state, a *stimulus* that triggers the functioning of the device, and *provided* that specifies the environmental conditions necessary for the functioning of the device. In a specific schema for function, the slots *given*, *stimulus*, and *provided* may not be filled. The *stimulus* in a function schema is a primitive to capture the device's interaction with the environment external to the device. An example of *stimulus* is Force on Switch in the representation of the function of a flash-light circuit.

Formally, in this work, a function *F* is a 6-tuple with the elements: type (the only type of functions considered here are of transformation type, indicated by *ToMake*, where a device



Note: All locations are with reference to components in this design.
All labels for states and transitions are local to this design.

Figure 3.4: Function and Behavior of Sulfuric Acid Cooler

functional specification:

```
(makes:output behavioral state
 {given: input behavioral state }
 by: causal sequence of behavioral states
 {stimulus: event in the external environment triggering the functioning of the device }
 {provided: conditions external to the device necessary for its functioning })*
```

Figure 3.5: Functional Specification Schema

transforms an input state into an output state), input state, output state, an internal causal behavior of this function, an external stimulus on which the function depends, and zero or more external conditions necessary for the function. That is,

$$F = (Type(ToMake), \{State_{IN}\}, State_{OUT}, B_F, \{Stimulus\}, \{Provided\}^*)$$

where $\{State_{IN}\}$, $\{Stimulus\}$, and $\{Provided\}$ are optional.

3.1.2.1 Behavioral State

The behavioral states of a device can be characterized by the state variables whose values may be transformed causally. A behavioral state (input, output, or intermediate states of an internal causal behavior) can be of two types: component state ($State_{COMP}$) which concerns a component in the device and substance state ($State_{SUB}$) which concerns the state of a substance in the device. Figure 3.6 shows the schema for a behavioral state in IDEAL. Both these types of states contain links to previous state, next state, preceding transition, and succeeding transition, and a component-state schema or a substance-state schema.

The schema for a component state specifies the component under description and its mode of operation in that state. The component schema itself consists of several slots such as its *is-a* and its *parameters* (i.e., the properties of components and their values) as explained earlier in the description of the structure. An example of a component state is the state of **switch** when its mode is **closed**.

The schema for a substance state specifies a partial description of the state of a substance at a particular point in the device topology. It consists of the slots for the location of a main substance, the schema for the main substance, and any substances contained in the main substance. The schema for a substance itself consists of *is-a* and a *PropertyList*. The schema for a specific substance Nitric Acid is shown in the dashed box in Figure 3.10. In IDEAL's memory of substances, *is-a* is a pointer through which a specific substance is linked to a general substance. Substances are described in more detail in a later section. In the description of a substance in a behavioral state, only some of the characteristic properties of the substance are of interest, and thus only their values will be specified.

Formally, the behavioral state is specified as follows:

behavioral state: (previous: previous state
 next: next state
 enabled-by: preceding state-transition
 enabling: succeeding state-transition
 substance-state-schema: substance description at current state:
 location
 main-substance: the schema for the substance:
 is-a
 (property value unit)*
 {contained substances' description }
 OR
 component-state-schema: component description at current state:
 component: the schema for the component:
 is-a
 (parameter value unit)*
 mode)

Figure 3.6: Behavioral State Schema

$$State = State_{SUB} || State_{COMP}$$

$State_{SUB}$ is a 3-tuple consisting of the location of a main substance L , the schema for the main substance Sub , and zero or more of substances contained in the main substance $Contained-Sub$. That is,

$$State_{SUB} = (L, Sub, Contained-Sub)$$

$$Contained-Sub = \{Sub\}^*$$

The schema for a substance Sub is a pair consisting of *Is-a* (e.g., Nitric Acid *is-a* liquid) and a *PropertyList*. A *PropertyList* is one or more of triplets of Property, its Value and Units (e.g., the property *temperature* of Nitric Acid with a value of $T1$ degrees). That is,

$$Sub = (Is-a, PropertyList)$$

IDEAL's knowledge of substances in a specific device is linked to its general knowledge of substances via *is-a*. IDEAL's memory of substances is organized in a *is-a* hierarchy as shown in Figure 3.10. Note that in Figure 3.10 many properties for a substance Nitric Acid are shown as <no-specification> which means that IDEAL knows that these properties are relevant to this substance but in its general knowledge there are no values specified for these properties. However, a specific substance, for instance, Nitric Acid in the low-acidity NAC can have some more values specified (i.e., acidity: low; temperature: T_1).

$State_{COMP}$ is a pair consisting of the schema for a component $Comp$ and its *Mode* in that state:

$$State_{COMP} = (Comp, Mode)$$

3.1.3 Behavior

The internal causal behaviors of a device are viewed as sequences (including cycles) of alternating *state transitions* between *behavioral states*. Annotations on the state transitions express the *causal*, *structural*, and *functional context* in which the transformation of state variables, such as substance, location, properties, and values, can occur. The causal context provides *causal relations* between the variables in preceding and succeeding states. The structural context specifies different kinds of structural information such as substances, components, structural relations among components and substances, and spatial locations in the device. The functional context indicates which functions of components in the device are responsible for the transition. Figure 3.4(b) shows the causal behavior that explains how **water** is heated from **temperature** t_1 to t_2' . $State_6$, the preceding state of *transition*₆₋₇, describes the state of water at location p_5 and $state_7$, the succeeding state, at location p_6 . The different types of annotations on *transition*₆₋₇ indicate the different types of context under which the transition can occur. For example, the annotation USING-FUNCTION in *transition*₆₋₇ indicates that the transition occurs due to the behavior **allow** of H_2SO_4 -pipe.

Note that the function of a device in IDEAL is an abstraction over an internal causal behavior of the device, that is, the initial state and the final state in an internal causal behavior are

respectively the input state and the output state in the function (i.e., an observable, output behavior of the device). For instance, the input state and the output state of the “Heat Water” function of Sulfuric Acid cooler (Figure 3.4(a)) are respectively the same as the initial state ($state_6$) and the final state ($state_7$) in the internal causal behavior “Heat Water” of the Sulfuric Acid cooler (Figure 3.4(b)). However, a designer may not *intend* every output behavior of a device as a desired function of the device. For instance, in a Nitric Acid cooler device, an abstraction over the internal causal behavior “Heat Water” may not be intended as a function of the device; instead, an abstraction over the internal causal behavior “Cool Acid” may be intended as its function. Thus, the functions of a device are a subset of its output behaviors which were actually intended by its designer.

A model M in a design case (or analogue) is an n -tuple consisting of a causal behavior B_F that delivers the function F in the case and zero or more behaviors exhibited by the structure S in the case. Like states can be of two types, causal behaviors can also be of two types corresponding to the types of states the behaviors contain. For instance, a causal behavior B_{SUB} is a sequence of alternating states ($State_{SUB}$) and state transitions ($Trans_{SUB}$).

$$\begin{aligned}
 B &= B_{SUB} || B_{COMP} \\
 B_{SUB} &= \{State_{SUB}Trans_{SUB}\}^+ State_{SUB} \\
 B_{COMP} &= \{State_{COMP}Trans_{COMP}\}^+ State_{COMP}
 \end{aligned}$$

3.1.3.1 Behavioral State Transition

A behavioral state transition is a partial description of a transformation of some device element during the functioning of the device. Figure 3.7 shows the schema for representing such a transformation in the SBF language. In addition to the links to the previous and next states, the behavioral state transition schema contains the slots *by-behavior*, *using-function*, *as-per-domain-principle*, *parameter-relations*, and *conditions* of different kinds that need to hold good in order for the transition to occur.

state-transition:

```

(previous_state: preceding state
next_state: succeeding state
{ by-behavior: pointer to a more detailed behavior explaining the transition }
{ using_function: component's function }*
{ as-per-domain-principle }*
{ parameter-relations }*
{ condition }*)

```

Figure 3.7: Behavioral State Transition Schema

A behavioral transformation of a device element may be explained at several levels of abstraction and detail. Thus, the state-transition schema may include a pointer to another behavior (i.e., *by-behavior* slot) which explains in greater detail the transformation described by that transition. The *by-behavior* pointer results in the hierarchical organization of the device internal behaviors.

In addition to pointing to a more detailed behavior, a state transition may explain a behavioral transformation in terms of the functions of structural elements of the device (i.e., *using-function* slot), or in terms of a domain principle (i.e., *as-per-domain-principle* slot) such as the physics laws (e.g., the law of conservation of momentum). The *using-function* slot of a behavioral state transition schema is filled with a list of schemas each of which refers to a component in the device and a primitive function of that component. A partial set of primitive functions consists of *allow*, *pump*, *create*, and *destroy*.

Moreover, the transition schema may be annotated with qualitative equations (i.e., *parameter-relations* slot) describing the changes to the values of different substance properties and component parameters because of the transition. Qualitative equations may be based on physics principles, but they are specific to the device parameters. The *parameter-relations* slot of the state-transition schema is filled with a list of qualitative equations, where each qualitative equation itself consists of a qualitative relation between values of two substance properties or between values of a substance property and a component parameter. A qualitative relation is an enumerated type and can currently have one of the two values: *directly-proportional-to* and *inversely-proportional-to*. In addition, the SBF representations have been extended to include a specification of quantitative equations involving simple operators (addition, subtraction, division, multiplication, and exponent). These equations are useful in enabling simple quantitative simulations.

Often, the occurrence of a state transition in a device behavior is conditioned upon the co-occurrence of other behavioral states in the device (a pointer to the state via *under-condition-state*), or the co-occurrence of other state transitions (a pointer to the transition via *under-condition-transition*), or specific structural relations among the device elements (a list of structural relations via *under-condition-structure*), or specific property values of a substance (a pointer to the partial description of the substance via *under-condition-substance*), or specific parameter values of a component (a pointer to a partial description of the component via *under-condition-component*). Thus there can be five different types of conditions described in a state transition schema in the SBF language.

A transition between two substance states $Trans_{SUB}$ is a 5-tuple consisting of zero or more qualitative equations (*Qual-Equation*), zero or more principles (*Principle*), zero or more conditions (*Condition*), zero or more functions of components (*Using-F*), and an optional behavior (*B*) all of which form different kinds of context under which the transition can occur.

$$Trans_{SUB} = (\{Qual-Equation\}^*, \{Principle\}^*, \{Condition\}^*, \{Using-F\}^*, \{B\})$$

Similarly, a transition between two component states $Trans_{COMP}$ is specified as:

$$Trans_{COMP} = (\{Principle\}^*, \{Condition\}^*, Stimulus)$$

A qualitative equation *Qual-Equation* is a 3-tuple consisting of a qualitative relation *Relation* between values of two substance properties or between values of a substance property and a component parameter. *Relation* can be one of the two enumerated types, *directly-proportional-to* and *inversely-proportional-to*. In general, there may be other types of relations but in the class of devices we have dealt with, they were sufficient. A *Qual-Equation* captures a qualitative relationship between two properties of a substance, or between properties of different substances, or between properties of substances and parameters of components. Whereas a *Principle* refers to domain principles such as physics laws (e.g., conservation of momentum). That is,

$$Qual-Equation = (Relation, LHS, RHS)$$

$$Relation \in \{ directly-proportional-to, inversely-proportional-to \}$$

$$LHS = (Sub, Property, Value)$$

$$RHS = (Sub, Property, Value) || (Comp, Parameter, Value)$$

Using-F in a transition is a pair consisting of a component *Comp* in the device and a primitive function *FP* achieved by that component. An *FP* can be one of the four enumerated types: *allow*, *pump*, *create*, and *destroy*.

$$Using-F = (Comp, FP)$$

$$FP \in \{allow, pump, create, destroy\}$$

A *Condition* under which a transition can occur can be of five types: a condition on a structural relation (*Condition_{STRUCT}*), a condition on a substance property (*Condition_{SUB}*), a condition on a component parameter (*Condition_{COMP}*), a condition on the existence of a state (*Condition_{STATE}*), and a condition on the occurrence of a transition (*Condition_{TRANS}*). That is,

$$Condition =$$

$$Condition_{STRUCT} || Condition_{SUB} || Condition_{COMP} || Condition_{STATE} || Condition_{TRANS}$$

where each of these conditions is a tuple as shown below.

$$Condition_{STRUCT} = (Struct-Relation, Comp, \{Comp\}^+)$$

$$Struct-Relation \in \{ contains, includes, serially-connected, parallelly-connected \}$$

$$Condition_{SUB} = (Sub, Property, Value)$$

$$Condition_{COMP} = (Comp, Parameter, Value)$$

$$Condition_{STATE} = (State)$$

$$Condition_{TRANS} = (Trans)$$

In the above structural relationships, *contains* and *includes* are different in that the former specifies a relationship between a component and a substance while the latter specifies a relationship between two components. The relations *serially-connected* and *parallelly-connected* are also different: the former specifies a relationship between components such that the output of one becomes the input to the others, while the latter specifies that the two components share the same input and the same output.

For instance, in *transition₆₋₇* (shown in Figure 3.4(b)), the UNDER-CONDITION-

STRUCTURE annotation specifies that the behavior **allow** of H_2SO_4 -pipe can allow the flow of heat only if the H_2SO_4 -pipe CONTAINS Sulfuric Acid with a temperature of T_1 that is greater than t_1 . The qualitative parameter relations on the substance properties, such as those shown for temperature in Figure 3.4(b), are a crucial part of describing the causal process underlying a transition. Annotations may also include conditions on other transitions as indicated by UNDER-CONDITION-TRANSITION. For example, *transition*₆₋₇ refers to another transition in which the temperature of H_2SO_4 changes from T_1 to T_2' . In addition, a transition may be annotated by the knowledge of deeper domain principles and qualitative equations.

The state transitions, in addition to being causal transitions, capture an implicit temporal ordering of events in the device functioning. Since, in general, the cause temporally precedes the effect, the antecedent state temporally precedes the consequent state. Moreover, the conditions on the transition implicitly capture temporal co-occurrence, i.e., if two state transitions are dependent upon each other, then they occur at the same time.

3.2 Primitive Functions

Bylander and Chandrasekaran (1985) have proposed a few primitive functions, such as *allow*, *pump*, and *create*, for the domain of physical devices, though not intended as a complete set. Unlike any higher-level function, primitive functions cannot be further decomposed into any subfunctions or associated internal behaviors, and thus a primitive function is also a primitive behavior.

A primitive function, like any other function, is represented in terms of an input state, an output state, and a set of behavioral requirements under which such a transformation is possible. For example, Figure 3.8 shows a representation of the primitive function **allow**. KRITIK (Goel, 1989) has primitive functions such as *allow*, *pump*, *create*, and *destroy* for the domain of physical devices. The functional context of transitions in behaviors of devices index into these primitive functions. Thus these primitive functions are the building blocks to compose higher level functions. Govindaraj (1987), in his Qualitative Approximation Methodology to model large dynamic systems, has proposed primitives to describe components in a system; but those primitives are *primitive components* and not primitive functions. However, the primitive components are closely related to "primitive" functions as they are responsible for the basic functions performed by the components. For example, the primitive *conduit* from his set of primitives has a function similar to the primitive function *allow*.

Primitive functions are also useful to classify functional differences between designs so that classes of designs can be discriminated at the top-level in a functionally organized memory of design analogues (explained later in Chapter 4).

Dealing with a variety of devices from new domains and mechanisms such as feedback and feedforward in this research has necessitated us to introduce two new primitive functions in the SBF theory: *transform* and *sense*. *Transform* is used to describe the behavior of a component

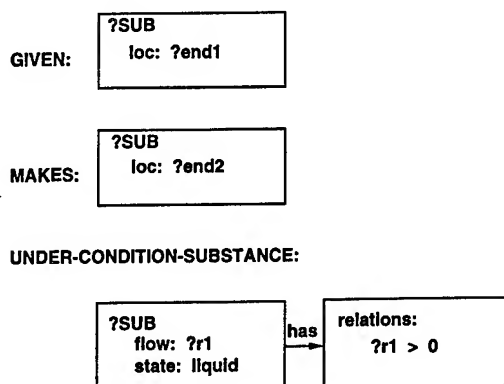


Figure 3.8: Primitive Function (and Behavior) ALLOW

that causes a transformation in a property value of an input substance when such change cannot be described by any of the other primitives. It is because the other primitives capture semantics of changes in specific properties of substances—for instance, the primitive *allow* specifies changes only in the location *loc* of a substance.² *Sense* is used to describe the behavior of a component that generates a signal (or some substance, in general) proportional to fluctuations in the value of some property of an input substance. The primitive function *sense* is necessary to capture the functioning of a class of devices that have feedback control or feedforward control mechanisms.

3.3 Substance Knowledge

Since the function of a device can be specified as a pair of input and output states of a substance in the device, general knowledge of substances is essential and useful in determining partial matches between functions during the retrieval of an analogue. In other words, it helps to index and organize design analogues hierarchically based on substances. In our theory, each substance is represented as a schema that consists of a name, its category information (i.e., *is-a*), and properties and corresponding values (default or ranges). To address the issue of what should be represented in a substance, we take a functional approach, that is, represent only that much required by the class of tasks and the domains considered. An analysis of some specific designs in the domain of physical devices suggests certain properties for specific substances; only those properties are included in the representation. For example, an analysis of designs of reaction-wheel assembly indicates that the properties of the substance *angular momentum*, namely, *magnitude* and *direction*, are important to describe their function.³ Hence

²In the current implementation of IDEAL we expanded the semantics of *allow* to include that of the new primitive *transform*.

³In component-substance ontology, things such as *angular momentum*, *heat*, and *electricity* which cannot be “seen” are also viewed as substances that can flow through components in realizing a behavior of a device. But in order to distinguish them from all those things that can be “seen” and/or that have one or more of the properties of shape, state of matter, and form, substances like *angular momentum* are categorized as *abstract substances*.

Angular momentum		Nitric Acid	
is-a:	abstract-substance	is-a:	acid
property-list:	property-list:		
magnitude:	---	state:	liquid
direction:	<positive or negative>	acidity:	---
		temperature:	---
		flow:	---

'---' indicates <no-specification>

Figure 3.9: Representation of Substances: Examples

a general representation of *angular momentum* should specify these properties. For example, the representations of two substances, *angular momentum* and *nitric acid*, are given in Figure 3.9. Note that in Figure 3.9 many properties for the substances Angular Momentum and Nitric Acid are shown as <no-specification> which means that IDEAL knows that these properties are relevant to this substance but in its general knowledge there are no values specified for these properties. However, a specific substance, for instance, Nitric Acid in low-acidity NAC may have some more values specified (i.e., acidity: low; temperature: T_1).

Substances are organized in a taxonomy (is-a hierarchy), with substances at the top-level divided into two categories: *abstract substances* and *concrete substances*. Figure 3.10 illustrates a partial memory of substances in IDEAL. Under the two top-level categories, substances are distinguished among them based on their properties. For example, concrete substances can have three subcategories based on their state of matter corresponding to *solid*, *liquid*, and *gas*. These distinctions are important in design-analogue adaptation because a component such as *pipe* can allow only liquids and gases to go through; hence the pipe can be substituted for a component only under certain substances. Further, liquids are divided into *acids*, *alkalis*, and *neutral liquids*. Abstract substances are divided into categories such as *angular momentum*, *heat*, and *electricity*.

Substances are indexed by specific substances used in design analogues. The representation of a specific substance in a design includes a pointer to the corresponding conceptual representation. For example, a specific substance Sulfuric Acid in the design of a high-acidity Sulfuric Acid cooler will have a pointer to the substance schema describing its general properties. Although this is a primitive way of indexing, it provides a way of linking conceptual information of substances and design analogues in which specific substances are used. This also helps in finding a substance for replacement during the adaptation of a design analogue.

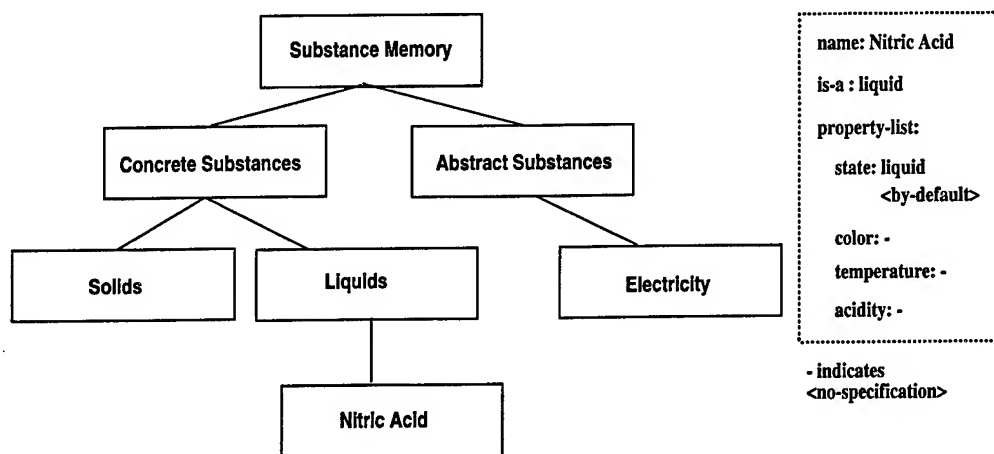


Figure 3.10: A Partial Memory of Substances in IDEAL

3.4 Component Knowledge

Components are the building blocks of structure of any physical device. Like some functions can be specified as pairs of substance states, some other functions may be specified as pairs of component states. Therefore the general knowledge of components is also useful in determining partial matches between functions during analogue retrieval. In our theory, each component is represented as a schema that consists of a name, its category information (i.e., *is-a*), its structural relations, its modality (i.e., modes of operation because a component can have multiple modes), its parameters and corresponding values (default or ranges), its functions, and its connecting points. For example, the representations of two components, namely, *switch* and *battery*, are given in Figure 3.11.

Structural-relations slot of a component specifies a list of structural relations between the component and the other components in the device. *Modes* in a component schema specifies one or more modes of operation of the component. *Parameters* contains a list of characteristic parameters of the component and the corresponding values and units. For example, the volume of a heat-exchange chamber has a value of 1 cu. ft. The *functions* slot of a component contains the set of primitive functions that the component delivers and the *connecting-points* specifies the structural points in the component where the other components can be connected. Note that the general representation of a component does not specify values to all these different slots. But a specific component in a device would have most of these slots specified.

Similar to substances, components can also be organized in a *is-a* hierarchy. Figure 3.12 illustrates a partial memory of components in IDEAL. In addition, components may also be functionally organized because in a design task, components are accessed and grouped together primarily based on the functions they deliver. Within a functional group of components, however, further distinctions may be drawn based on component categories. For example, electrolyte

Switch	Battery
name: switch	name: battery
is-a: control-device	is-a: electric-source
parameters:	parameters:
mode: <open or closed>	type: <electrolyte or Ni-Cd>
	voltage: <no-specification>

Figure 3.11: Representation of Components: Examples

batteries and nickel-cadmium batteries have the same function (i.e., *pump* electricity) and are batteries. Within electrolyte batteries, there can be subclasses based on voltage they deliver such as low-voltage electrolyte batteries and high-voltage electrolyte batteries (or, 1.5-volt electrolyte batteries, 3-volt electrolyte batteries, and 9-volt electrolyte batteries).

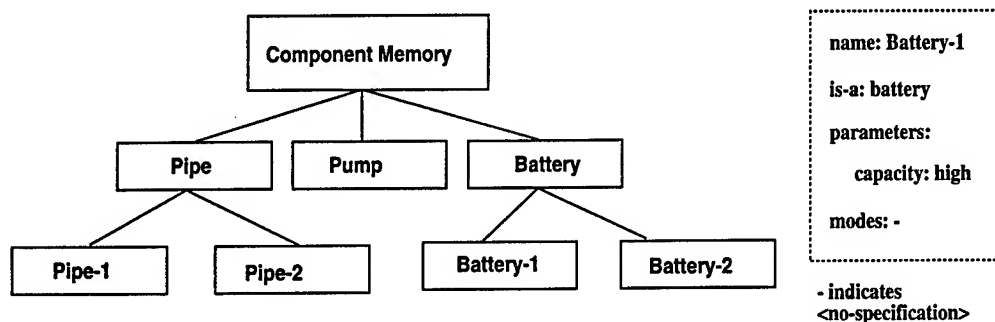


Figure 3.12: A Partial Memory of Components in IDEAL

Like substances, components can also be indexed by specific components used in the design analogues. The representation of a specific component in a design includes a pointer (via *is-a*) to the corresponding conceptual representation. This indexing scheme coupled with functional indexing for components helps in some adaptation processes, for example, in structure modification due to the instantiation of component-replacement. An appropriate new component to replace a component in a design analogue can be found by accessing the general knowledge of a component whose instance the current component is.

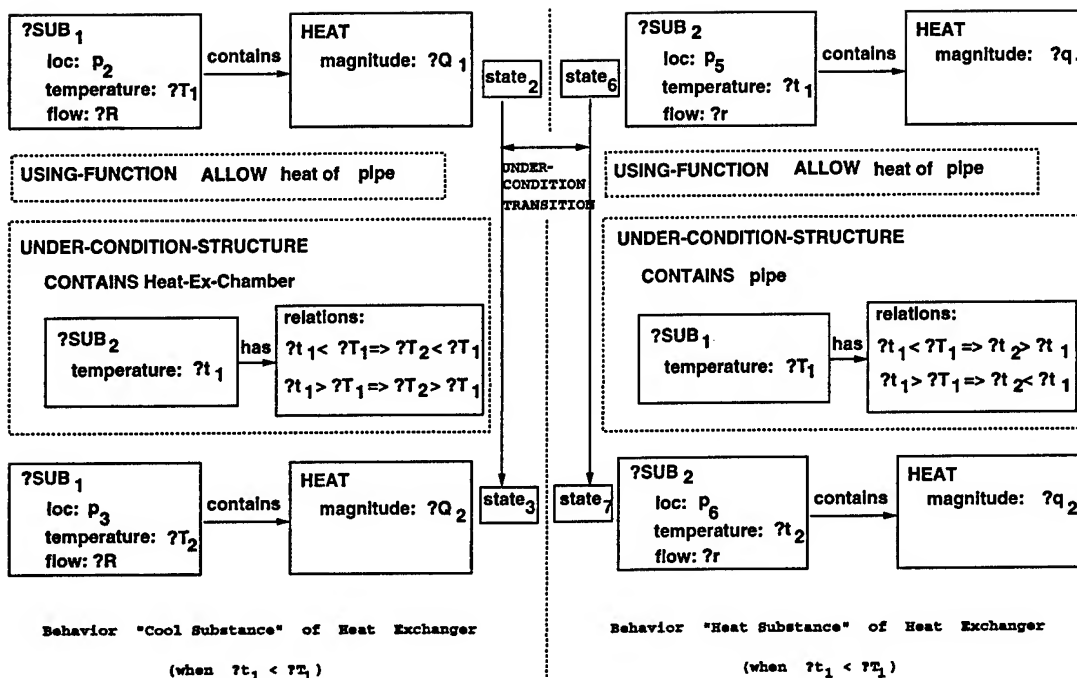
3.5 Structure-Behavior-Function Models of Design Patterns

Design patterns, in general, capture different kinds of relationships between design elements, such as spatial, temporal, functional, and causal relationships. Design patterns are generic and case-independent because of the nature of the relationships and the level of abstraction of those relationships. They can be of different types depending on the kinds of relationships they capture. For instance, design patterns can be spatial patterns in certain domains (e.g., the design domains of pictures and buildings) because they capture spatial relationships. Or they can be temporal patterns in some other domains (e.g., the domains of designing symphonies and music), or functional and causal patterns in yet other domains (e.g., the domains of designing physical devices such as electric circuits and heat exchangers). In this research, we focus on the functional- and causal-type design patterns because the task context is design of physical devices.

The functional- and causal-type design patterns themselves may be of different types. We focus on two specific types of design patterns, namely, generic physical processes (GPPs) and generic teleological mechanisms (GTMs), and provide a content theory for them *in terms of* the SBF language. The same SBF language we used to represent device models can be used to represent models of GPPs and GTMs. Although the SBF language provides primitives for the physical structure of devices, the functions of devices, and the internal causal behaviors of devices, since design patterns capture only patterned relationships between design elements, the SBF representations of design patterns are devoid of information about devices' specific physical structure. However, those representations still capture the *causal structure* in the behaviors of the classes of devices. Hence, the function and behavior aspects of the SBF language are especially useful for representing the GPPs and GTMs. We will describe their SBF representations in the following two sections.

3.5.1 Generic Physical Processes

A Generic Physical Process captures causal relationships between the output behaviors and the internal behaviors of physical devices. Depending on the level of abstraction of the specific substances and components in these relationships, GPPs can be represented at different levels of abstraction. For instance, at lower levels of abstraction, the causal relationships in a GPP may refer to prototypical structural elements that characterize the class of devices to which the GPP applies. An example of such a GPP from IDEAL is the SBF model of a heat exchanger (Figure 3.13) that is general enough to provide explanations of how a cooling device works as well as how a heating device works. Thus the primary characteristics of a lower level GPP are as follows:



Note: Symbols prefixed with ? denote variables.
 All locations are with reference to components in this design.
 All labels for states and transitions are local to this design.

Figure 3.13: An SBF Model of a Heat Exchanger

- Overall structural topology can be same as in its specializations but there is no information of specific components (instead of specific components, classes of components may be described as in, for example, Figure 3.13—note that the component *pipe* is not a specific component in a device but rather is a prototype component).
- The function is more general than in its specializations (for example, the range of transformation in a property is higher than in the specializations, or parameters in the function are variablized, that is, variables are substituted for the values of properties).
- At least one of the several types of context in a transition in the causal behavior of the device is more general than the context in its specializations (for example, in Figure 3.13, relations between the values of *temperature* cover changes in both directions, that is, increase in *temperature* and decrease in *temperature*).

Note that the representation of the GPP of heat exchange process shown in Figure 3.13 uses the same primitives as described earlier for the SBF models of devices, such as behavioral states, state transitions and the different types of annotations for transitions. But instead of referring to specific substances and property values, this representation specifies variablized or parameterized substances, property values, and conditions. This representation of the GPP specifies that the transformation of the temperature of some substance *?sub₁* flowing with some flow rate *?R* from a location *p₂* to another location *p₃* is mutually dependent on the transformation of the temperature of some other substance *?sub₂* flowing with some flow rate *?r* from a location *p₅* to another location *p₆*. The direction of the transformation in one substance is opposite to that in the other as indicated by the relations on substance temperatures annotating the transitions. In addition, it specifies (using the UNDER-CONDITION-STRUCTURE annotations) the structural relations that need to hold good in order for the two behavioral state transitions to occur.

GPPs at higher levels of abstraction typically encapsulate causal processes that underlie a larger class of devices and describe physical principles. They explain how certain properties of substances and parameters of components undergo transformation or how they are maintained in a particular state without referring to any specific structural information. Examples of such GPPs are the process of *heat flow* and the process of *electric flow* which are also abstract descriptions of behaviors, often associated with a corresponding physical principle. That is, the Heat-Flow GPP is essentially the behavior that the zeroth law of thermodynamics epitomizes; the Electric-Flow GPP is the behavior that Ohm's law epitomizes. The SBF model of the Heat-Flow GPP shown in Figure 3.14 is also a representation of the physical principle of the zeroth law of thermodynamics. Note that, in general, physical processes may have more than one causal transition in their descriptions. This representation of the Heat-Flow GPP specifies that the transformation of the temperature of some substance *?sub₁* is mutually dependent on the transformation of the temperature of some other substance *?sub₂* that is in thermal contact

with $?sub_1$. The direction of the transformation in one substance is opposite to that in the other as indicated by the relations on substance temperatures annotating the transitions. Note also that unlike the lower-level GPP of heat exchange, the Heat-Flow GPP does not refer to the flow of the substances $?sub_1$ and $?sub_2$, nor does it refer to the change of locations, because the Heat-Flow GPP is a higher-level abstraction.

Physical principles such as the zeroth law of thermodynamics and Ohm's law are abstract and cryptic descriptions of behaviors that a large class of devices exhibit and obey. For example, the behavior of a cooling device includes a specialization of the zeroth law of thermodynamics and the behavior of a simple electric circuit obeys Ohm's law. Physical principles typically capture relations and dependencies between properties of substances and parameters of components without referring to any specific structural information. For example, Ohm's law relates the properties voltage, current, and resistance that certain classes of substances have and that may be delivered by some functions of components in the domain of electrical devices.

3.5.2 Generic Teleological Mechanisms

GTMs capture functional and strategic relationships between differences in functions of devices (i.e., a subset of output behaviors) and differences in their internal causal behaviors. A few examples of GTMs are cascading, feedback, feedforward, and device composition. GTMs are *teleological* in that they result in specific functions. For example, the cascading mechanism takes as input the desired function and the function (with a lesser range) of any single component available and suggests a behavioral pattern where the behaviors of several components of the lower-range functionality are replicated in a *functional additive manner* (that is, they are composed such that the overall function is a sum of the individual functions) and which results in the desired higher-range function. These mechanisms are *generic* in that they are device independent. The cascading mechanism, for example, can be instantiated in any specific device that satisfies its applicability conditions (that is, the change in a property value is additive with respect to the replication of a device that delivers a smaller change).

Since GTMs do not refer to any specific physical structure, they are also represented using the behavior and function aspects of the SBF representations. The SBF representation of a GTM encapsulates two types of knowledge: knowledge about the patterned difference between functions of known designs and desired designs that the GTM can help to reduce, and knowledge about modifications to the internal causal behaviors of the known designs that are necessary to reduce this difference. For example, Figure 3.15 shows an SBF model of the cascading mechanism from IDEAL: Figure 3.15(a) illustrates the former type of knowledge for the cascading GTM and Figure 3.15(b) illustrates the latter type of knowledge for the GTM. The behavior modification that the cascading GTM suggests is to replicate end-to-end the internal causal behavior of the known design from the initial state of the desired function n times and form a goal to achieve the residual transformation, the behavior of which needs to be composed at the end state of

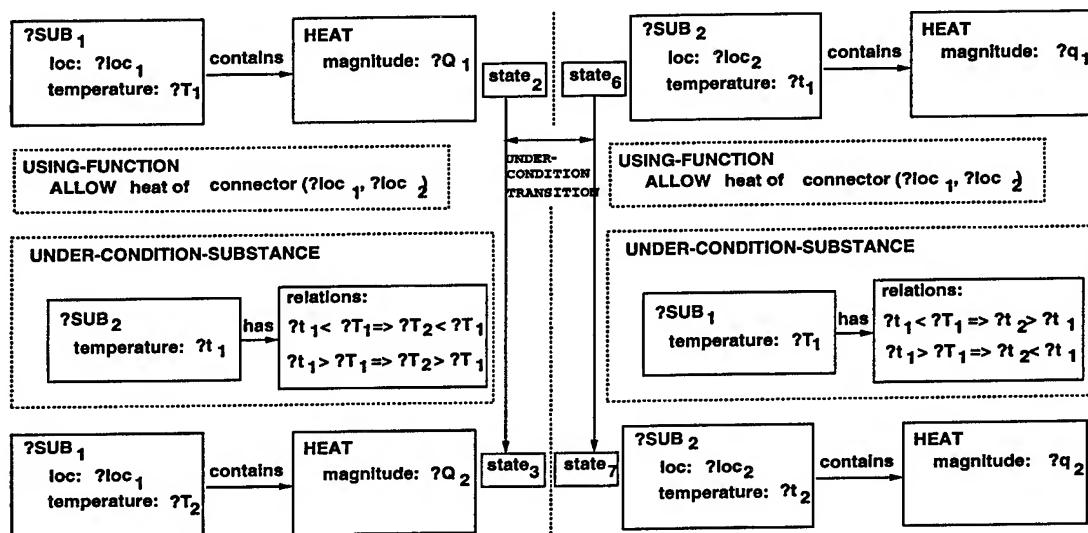


Figure 3.14: An SBF Model of the Heat-Flow GPP (i.e., The Zeroth Law of Thermodynamics)

the replicated segment. The number of replications n is the integer dividend of the ratio of the desired functional transformation to the functional transformation achieved by the known design (i.e., $n = \lfloor \text{desired functional transformation} / \text{candidate functional transformation} \rfloor$). Note that this representation captures only the serial cascading of causal behaviors in order to achieve larger transformations.

DESIRED DESIGN:

GIVEN: **?SUB**
?prop1: ?val12

MAKES: **?SUB**
?prop1: ?val22

BY-BEHAVIOR: Behavior B2

CANDIDATE DESIGN:

GIVEN: **?SUB**
?prop1: ?val11

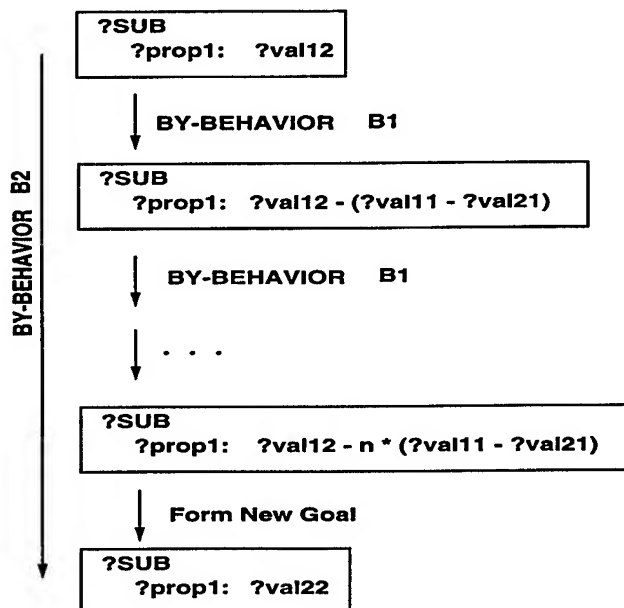
MAKES: **?SUB**
?prop1: ?val21

BY-BEHAVIOR: Behavior B1

CONDITION:

$?val22 \sim ?val12 \gg ?val21 \sim ?val11$

(a) Functional Difference the Cascading Mechanism Reduces



(b) Behavior Modification the Cascading Mechanism Suggests

Figure 3.15: A Complete Description of the Cascading Mechanism in SBF Representation

CHAPTER IV

ANALOGUE MEMORY: MODEL-BASED INDEXING, ORGANIZATION, RETRIEVAL AND STORAGE

The theory of MBA raises three sets of issues concerning the analogue memory. The first set of issues concern the indexing and organization of analogues in memory. The indexing and organization of analogues is important precisely because the successful retrieval of a right analogue is closely tied with the way analogues are indexed and organized in memory. The issues here are: How might the design analogues be indexed in memory? Whether they need to be indexed in one way or multiple ways? How might they be organized in memory? The overall task (such as design) for which a source analogue needs to be retrieved partly determines the nature of indices for the analogues in memory because the task really determines what is specified in the problem and in turn what is available to match with the indices. Similarly, the types of information specified in problems determine whether the analogues need to be indexed in multiple ways.

The second set of issues relate to the retrieval of analogues from memory. Given a target problem and a set of source analogues, what are the processes of retrieval? What kind of features in the target problem might determine the retrieval of analogues? What are the criteria for measuring similarity between a problem and a design analogue in memory? Since retrieval is the first step in model-based analogy, and an important one for the method of analogical reasoning, one of the issues is how can it be made efficient and effective? The retrieval of a "right" source analogue for a given problem is important for the success in subsequent steps in model-based analogy.

The third set of issues relate to storage of analogues in memory for later use. It is important to assimilate a new analogue with the other analogues in memory appropriately. The issues here are: How can the memory be dynamically re-organized when new design analogues are stored? Where do the indices for a new design come from? How might they be acquired automatically?

This chapter is organized in 3 parts: first, we describe the model-based indexing and multiple organizations of design analogues; then we describe the retrieval task and its subtasks; and finally, we describe how new design analogues are stored and how models help in learning indices for the analogues. Our descriptions will be at 3 levels of abstraction: english narrative, algorithms, and specific examples (in particular, sulfuric acid cooler and nitric acid cooler designs).

4.1 Indexing and Organization of Design Analogue Memory

In model-based analogy, the issues of indexing and organization of analogues in memory are important because the subtask of analogue retrieval is dependent on the particular indexing and organization. Given a new problem, the task of retrieving a similar analogue is to bring out the “best” matching analogue from memory. Since tasks can be sent as probes into memory, indexing schemes and methods of index learning depend on the functional requirements of the tasks if not specific to a task. Since in the context of design problem solving, a problem involves specifying functions desired of the new device and sometimes the structural constraints that the new design should satisfy, the stored design analogues in IDEAL are indexed both by their functions and structure. Thus it uses multiple types of indices for its design analogues. Furthermore, the indexing scheme thus reflects the reasoning tasks addressed in our computational process. IDEAL’s functional indexing scheme is similar to that in KRITIK (Goel, 1992b), but the latter does not index designs structurally. Also, KRITIK organizes its designs only in a flat memory, unlike IDEAL.

4.1.1 Functional Indexing and Organization

In order to enable successful, efficient and effective retrieval of analogues from memory, in IDEAL we organize them hierarchically along the specific types of indices. We chose the generalization-specialization relationship between the values of features in device functions as the principle of hierarchical organization because that supports the need of retrieving a source analogue that may match partially but as close as possible with the given functional specification in the target problem. Thus the design analogues in IDEAL are organized in generalization-specialization hierarchies. A generalization-specialization hierarchy of design analogues contains designs whose functional specifications are generalized at the higher levels and designs whose functional specifications are specialized at the lower levels in the hierarchy. As described in Chapter 3, a function in model-based analogy is expressed in terms of substance schemas. Since the substance schema specifies properties of substances, IDEAL uses them as dimensions along which design analogues are generalized/specialized. For example, designs of *acid coolers* are organized along the dimension of property *acidity*, and discriminated on the corresponding values *low vs high* as shown in Figure 4.1.¹ The *HNO₃ cooler* case in Figure 4.1 is a design of low-acidity nitric acid cooler and hence stored under the category that refers to low-acidity coolers.

The functions at the higher-level nodes are more general than those at the lower-level nodes in the sense that the values of the property (that is the dimension of generalization in this hierarchy) in the functions of design analogues associated with a node at a higher level subsume the values of the property in the functions of design analogues associated with a node at a lower

¹For instance, the property *acidity* is important because the choice of *pipe* in the design depends on whether it has to allow a low-acidity substance or a high-acidity substance.

level. For instance, the higher-level node **Acid-Coolers** in Figure 4.1 has both classes of designs, low-acidity coolers and high-acidity coolers, associated with it. In contrast, the lower-level node **Low-Acidity-Coolers** has only the designs of acid coolers with low acidity. Formally, the set of analogues associated with a node in the hierarchy is a superset (\supseteq) of those associated with any of its immediate child nodes.

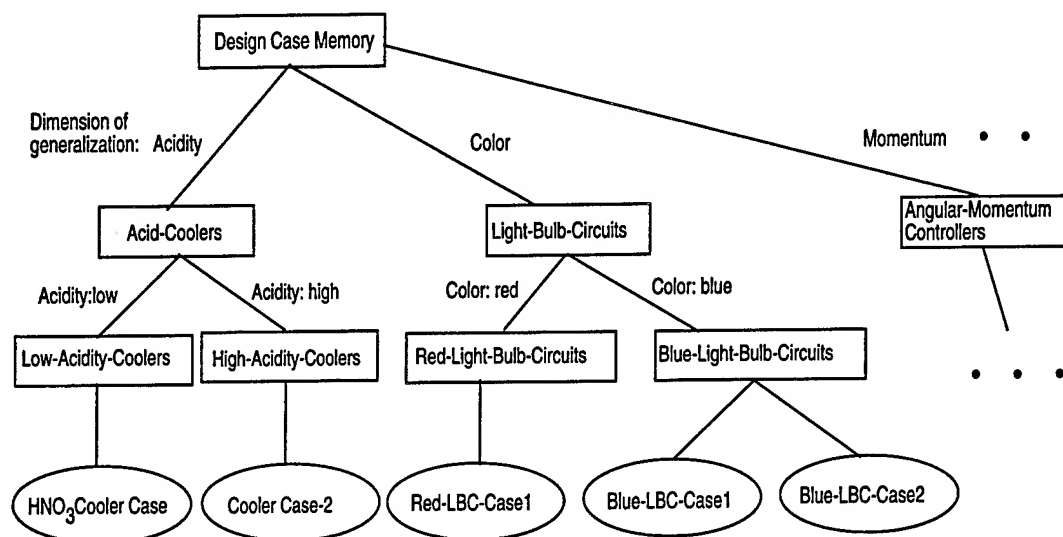


Figure 4.1: A snapshot of a functionally organized analogue memory

4.1.2 Organization Based on Primitive Functions

Figure 4.2 illustrates a snapshot of IDEAL's memory organized around primitive functions of devices at the top level of the hierarchy and along the properties of substances in device functions at the lower levels. This type of organization is useful in tasks where the probes specify the primitive functions of a desired design and not the functional specifications. That is, for instance, a design adaptation subtask of the process of MBA spawned by reasoning from the behavior of a source design could specify the primitive functions of a desired component (or in general, a desired substructure) to replace an old one in the source design. Suppose that the memory is organized around primitive functions (as illustrated in Figure 4.2). Since the desired functions in the input to the overall design task are specified in terms of input and output states (substance or component state schemas), retrieval from this type of memory incurs an additional inferential burden necessary to infer what may be the primitive functions that compose into the given function. Therefore, to support the retrieval tasks triggered from the different stages of MBA, it is desirable to organize analogues both by the primitive functions and by the device functions (i.e., substance properties in the input and output states of the device functions).

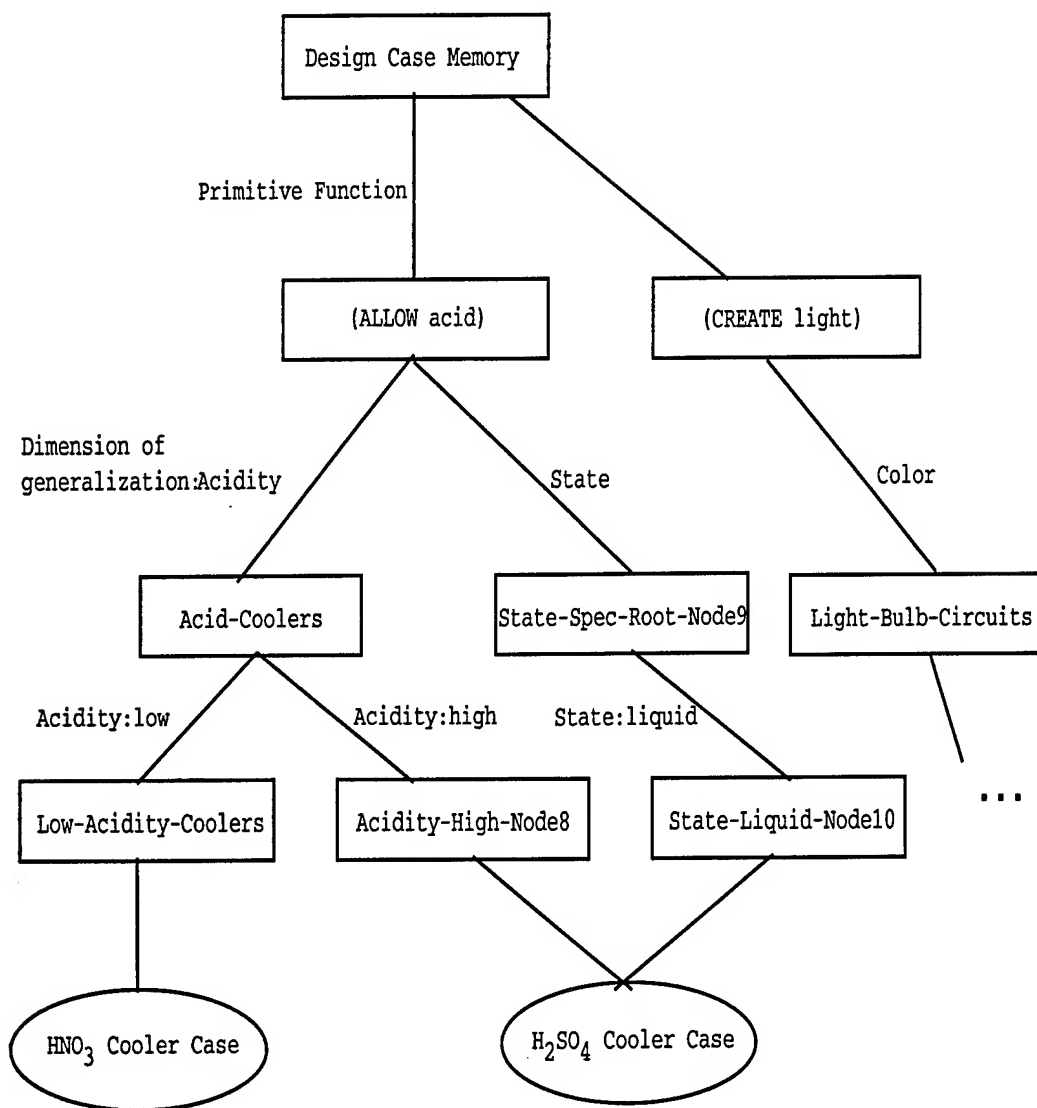


Figure 4.2: A Snapshot of A Memory Organized by Primitive Functions

4.1.3 Structural Indexing and Organization

The design problems in device design generally specify the functions desired of the new devices. But some class of problems may also specify structural constraints besides functional ones. The output of the design task for a problem in that class is the device structure that delivers the desired function and also satisfies the given structural constraints. Since the mapping between desired functions and the device structures (that can deliver the desired functions) is many-to-many in some domains, neither the function not the structural constraints alone are sufficient to retrieve a small set of source design analogues. Therefore in order to constrain the retrieval subtask and solve such class of design tasks by analogy, it is useful to index past designs by the structure of the devices in addition to their functions.

In device design, the structural constraints are specified in terms of the structural relations desired in the design, types of components (or substructures) to be used, and values or ranges desired for component parameters. Therefore in MBA we use the structural relations between different components in a design structure and the parameters of components in the design as dimensions of generalization-specialization for organizing design analogues structurally. For example, Figure 4.3 illustrates how *acid coolers* are organized in IDEAL along the dimensions of structural relations INCLUDES and CONTAINS at one level, and further discriminated between them based on values of component parameters. The HNO_3 cooler case in Figure 4.3(a) is stored under the structural relation (INCLUDES Heat-Exchange-Chamber HNO_3 -pipe-2) and along the parameter capacity of the pipe.

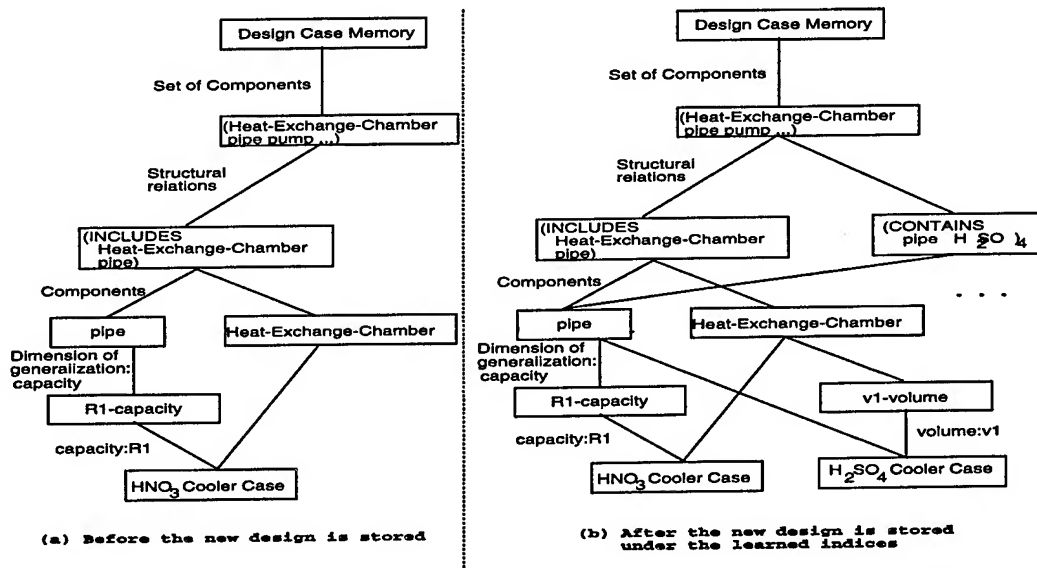


Figure 4.3: Snapshots of IDEAL's structurally organized analogue memory

4.2 Retrieval of Design Analogues

The first step in solving problems by model-based analogy is the retrieval of a best matching design analogue from the memory given the target design problem. This subtask is important in the computational process because the output of the retrieval partly determines whether a solution to the given problem can be generated or not. The desirable properties of a process for the analogue retrieval are that it is successful, efficient and effective. Some of the important issues in the retrieval of analogue are: how to index the source analogues in memory in order to enable quick and effective retrieval; what features in the target problem may be used to probe the memory of analogues; whether and how any additional features may be derived from what is specified in the target problem so that it facilitates retrieval of "a" source analogue (i.e., elaboration of the target problem); and when there are multiple partial matches, how they can be ordered.

The information available for the retrieval subtask to probe the memory is determined by the overall task being solved by the computational process and the indexing scheme used. If, for a given problem, no analogues can be retrieved then it results in an immediate failure of the process of MBA to solve the problem. The reasons may be that there is no design analogue in the memory that is relevant for the given problem by any means, or that the problem features and the indices are not of the same type or same level. In such situations, it is desirable to have a process that elaborates the given problem and transforms the features or derives other features because the features resulting from such a process may give rise to retrieving a source analogue. Under the condition that the retrieval of "a" source analogue, even if it becomes hard to adapt, is more desirable than a failure at retrieval, then a process of problem elaboration is useful.

When there are multiple analogues that match with the target problem, then there is a need for ordering them by some criteria so that the best analogue can be selected for adaptation. In MBA, we use a qualitative estimate of the ease of adaptation of the retrieved analogues for satisfying the requirements of the new design as the ordering criterion (Kolodner, 1989). The best matching source analogue is first selected for transfer & modification, and when using that analogue does not lead to a satisficing design for the target problem, then the next best matching analogue is selected, and so on.

The retrieval subtask thus has three further subtasks in our computational process of MBA: elaboration of design problems, selection of candidate design analogues, and ordering of candidate design analogues. We will illustrate these three subtasks in the following three sections with an example from IDEAL. Consider, for instance, the task of designing a device that delivers the function of cooling a high-acidity sulfuric acid. IDEAL accepts representations of target problems in the SBF language. The specification of the desired function in the SBF language is shown in Figure 4.4.

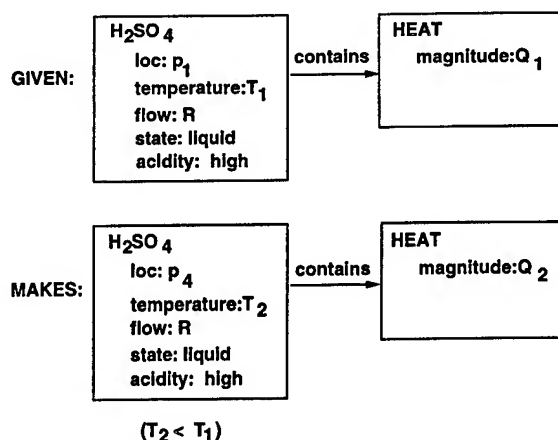


Figure 4.4: Function of Cooling High-Acidity Sulfuric Acid

4.2.1 Elaboration of Design Problems

The elaboration task takes as input a possibly incompletely specified problem. It has the goal of giving as output a more complete specification of the problem so that the given problem can be compared with the problems of past analogues. For instance, in design, it takes an incomplete functional specification and gives as output a functional specification with more properties and values specified.

Since the functions of devices may be specified as substance schemas, elaboration in design, for instance, could involve specifying more properties of the substances. General knowledge of substances in the domain (e.g., an is-a hierarchy of substances as described in Chapter 3) helps in deriving/implying unspecified properties for a substance. For example, even if the functional specification of sulfuric acid cooler (Figure 4.4) did not mention the property *acidity*, IDEAL could have elaborated the specification to include the property inferring the same from the knowledge that sulfuric acid *is* an acid. However, the values for the properties that it could infer would only be the default values (or no values) specified in the general knowledge of acids. Nevertheless, identifying the unspecified properties in this way helps in retrieving analogues that are stored under those properties. For instance, given the function shown in Figure 4.4 *without* any specification of acidity, IDEAL's elaboration process enables it to retrieve analogues organized in memory along the dimension of acidity.

4.2.2 Selection of Candidate Design Analogues

The selection task takes as input the specification of a target problem and gives as output a set of past design analogues whose problem specifications match at least partially with the target problem. Since a perfect match for a given problem may not always be available, we allow partial matches to be retrieved in the process of MBA. The selection of a candidate analogue is based on

its usefulness for solving the given problem. The analogues are selected such that the differences in the functions of the target problem and the selected analogues are in the known type of functional differences. For instance, in the SBF representation of device functions, two functions can differ only in their input states, or only in their output states, or in both. Furthermore, two substance states (input or output) from two different functions can differ in these ways: substances can differ (i.e., substance difference), the values of a common substance-property can differ (i.e., substance-property-range difference), the variations or fluctuations in the values of a common substance-property can differ (i.e., substance-property-value-fluctuation difference), a substance property may be specified only in one of the states (i.e., substance-property-unspecified difference and substance-property-additionally-specified difference). Similarly, two component states can differ in their respective properties and corresponding values.

In design task, the functional specification of the desired design is matched with the functions of the stored design analogues in order to retrieve any matching analogues. That is, the properties in the component-substance schemas in the input and output states of the functional specification of the desired design are matched with the properties in the component-substance schemas in the corresponding input and output states of functions of the design analogues stored in memory. If the functions of the desired design and a stored design analogue match at least partially, then the stored design analogue is judged as potentially useful for realizing the function of the desired design and is selected as a candidate analogue. Figure 4.5 shows the algorithm that IDEAL uses for selecting matching analogues. It searches through the functionally organized memory of analogues along the dimensions of generalization that correspond to the properties specified in the component-substance schemas in the desired function. Along each dimension of generalization, it goes as far specific as possible comparing the value of that property specified in either the input state or the output state of the desired function. It collects the design analogues associated with the most specific value it could reach along each dimension.

The matching of the functional specification of the high-acidity Sulfuric Acid Cooler (SAC; Figure 4.4) with the functions of the designs in the memory of analogues (Figure 4.1) results in the selection of the design for the low-acidity NAC (Figure 4.6). The design of low-acidity NAC is retrieved because of a partial match between its function and the function of high-acidity SAC along the dimension of acidity. Consider a hypothetical situation where analogue memory had both low-acidity NAC and a neutral-acidity motor-oil cooler (i.e., a design for cooling motor oil which has neutral acidity). In such a situation, IDEAL would have selected both these designs because both of them match the desired function partially on the property, acidity.

4.2.3 Ordering of Candidate Design Analogues

Since in the selection of candidate design analogues partial matches with the target problem are also selected, often there can be more than one candidate design analogue. And, when multiple candidate design analogues are selected, there is a need for ordering them so that the "best"

Input: • Desired function, $F_{desired}$.

Output: • A set of design analogues Set_{Cases} whose functions partially match $F_{desired}$.

Assumptions: • **root-list** contains the root nodes of all the hierarchies along with the dimensions of generalizations, and **dimension-list** contains all the dimensions of generalization.

Procedure:

```

SELECT( $F_{desired}$ );
begin
   $Set_{Cases} = \{\}$ ;
  selected-nodes =  $\{\}$ ;
  tobe-seen-nodes =  $\{node_i \mid node_i \in root-list \wedge$ 
    function associated with  $node_i$  and  $F_{desired}$  have at least one common
    property specified.  $\}$ 
  while not-empty(tobe-seen-nodes) do
    begin
      child-nodes = MATCHING-CHILDREN ( $F_{desired}$ , first(tobe-seen-nodes));
      if empty(child-nodes)
        then selected-nodes = first(tobe-seen-nodes)  $\cup$  selected-nodes;
        else tobe-seen-nodes = tobe-seen-nodes  $\cup$  child-nodes;
      tobe-seen-nodes = rest (tobe-seen-nodes);
    end;
   $Set_{Cases} = \{case_i \mid node_i \in selected-nodes \wedge case_i \text{ is associated with } node_i \}$ ;
  return ( $Set_{Cases}$ );
end.

MATCHING-CHILDREN( $F_{desired}$ , node-in-hierarchy);
begin
  child-nodes =  $\{\}$ ;
  foreach (dimension  $\in$  dimension-list) do
    begin
      child = DISCRIMINATE (dimension,  $F_{desired}$ ,
        get-children(dimension, node-in-hierarchy));
      if not-empty (associated-cases(child))
        then child-nodes = {child}  $\cup$  child-nodes;
    end;
  return (child-nodes);
end.

DISCRIMINATE(property,  $F_{desired}$ , children);
begin
  foreach (child  $\in$  children) do
    begin
      if (value-of(property, i/p-state( $F_{desired}$ )) = value-of(property, i/p-state(F(child)))
         $\vee$  (value-of(property, o/p-state( $F_{desired}$ )) = value-of(property, o/p-state(child)))
        then return(child);
    end;
  return (nil); /* as failure */
end.

```

Figure 4.5: The Selection Algorithm

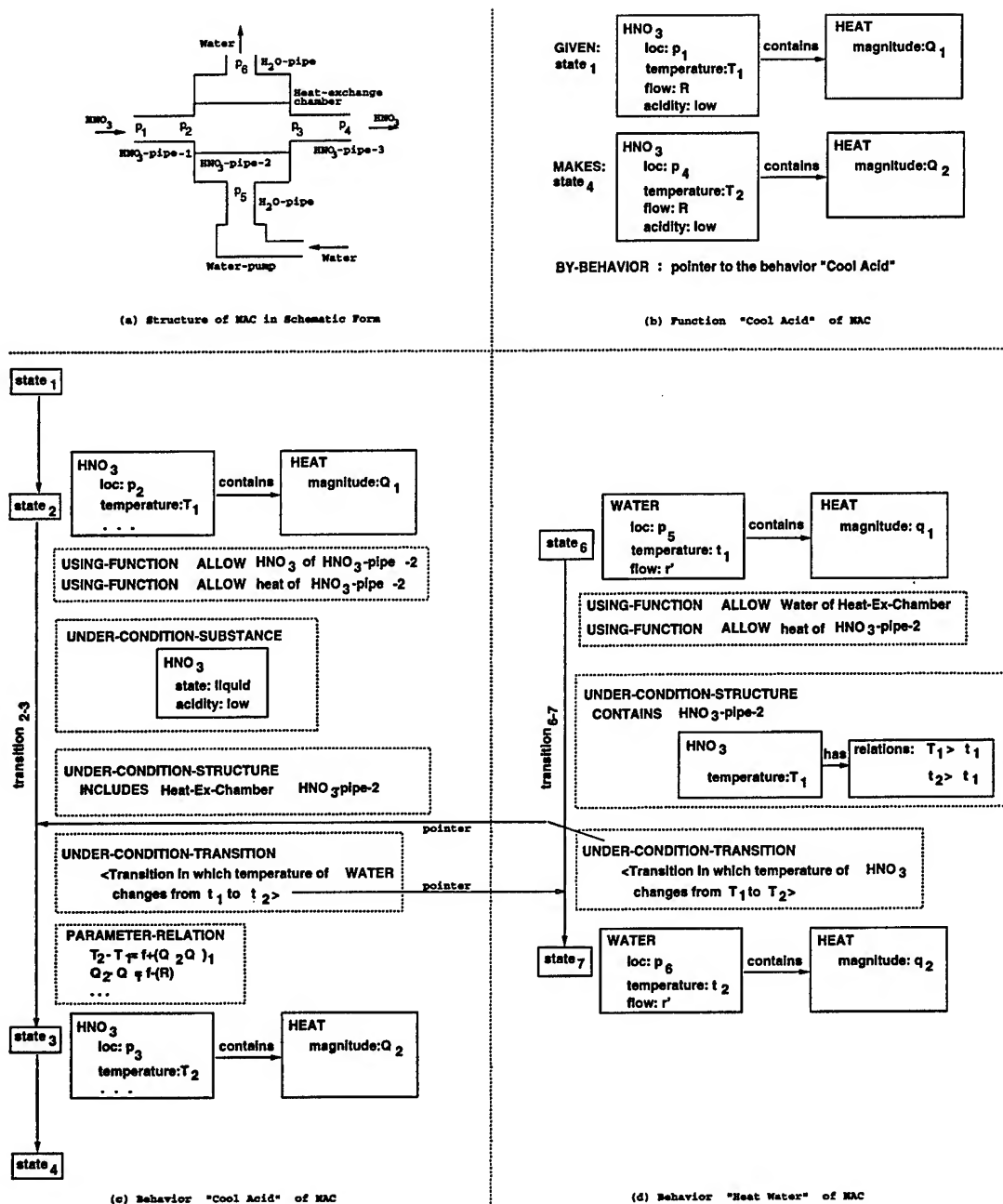


Figure 4.6: Design of Low-Acidity Nitric Acid Cooler

matching analogue based on whatever criteria used can be chosen as the source analogue for further processing. The ordering task thus takes as input a set of candidate analogues and gives as output an ordered set of analogues based on some measure of their ease of transfer and modification (i.e., adaptation). In MBA, the ordering of analogues is based on the ordering of differences between the target problem and the problems of the candidate design analogues.

Computing Differences between the Source and Target Problems: Computing the differences between a source design problem and the target problem requires that the two problem specifications be in a uniform representation and at the same level of abstraction. Otherwise, the comparison between the problems becomes hard or even impossible. In MBA, we use the SBF representations to specify the design problems. Since the SBF language provides primitives for representing a variety of design problems, and it is a well-defined and uniform language across different problems, identifying differences between problems in MBA is not computationally complex. The SBF language provides a taxonomy of functional differences between design analogues. The measures used in model-based analogy are based on a qualitative, heuristic estimate of how easy it might be to reduce the difference.

A Typology of Functional Differences: The SBF language provides a vocabulary for expressing certain types of functional differences between design analogues. Since the device functions are expressed in certain ways in the SBF language, there are a small number of dimensions along which two functions can differ; the differences can be many along each dimension. The typology of functional differences includes the categories of substance difference, substance-property-value difference, substance-location difference, component difference, and component-parameter difference (Goel, 1989). These differences are along the dimensions of all possible constituents of the representation of functions in SBF language. Modeling devices with the control mechanisms such as feedback and feedforward and the task of designing such devices in this work has brought out another important type of functional difference that we call *substance-property-value-fluctuation difference*.² That is, the difference between a desired function and the function in a retrieved design is in the range of fluctuation of a property value for the input or output substance. For instance, the difference between the function of a device without feedback mechanism and that of a device with feedback is in the fluctuation in a value of the output substance property such as *large* vs. *small* fluctuation.

There can be several different types of matches possible upon the retrieval of a set of candidate analogues: (1) the set contains an *exact match* in which case the target problem is solved; (2) the set is empty in which case the target problem cannot be solved by analogy; (3) the set is a singleton in which case there is no need for ordering (but the differences between the source and target problems are computed for use in the task of transfer and modification); and (4) the set contains multiple partial matches in which case the ordering is performed.

In the current example (i.e., retrieving based on the functional specification shown in Fig-

²Thus, this list is not intended as an exhaustive or complete set of functional difference types for the domain of physical devices.

ure 4.4), the situation is *single partial match* because the stored design of Nitric Acid Cooler is the only design analogue that matches partially with the desired function of high-acidity sulfuric acid cooler. But, for the purpose of illustrating how IDEAL orders analogues, consider a different situation where two low-acidity cooler designs, a low-acidity NAC and a low-acidity Sulfuric Acid cooler, were selected.

IDEAL adopts its metrics of ordering from KRITIK. The measure of how “large” is a functional difference is dependent on the following two aspects: *matching of behavioral states* and *matching of behavioral state features*. IDEAL’s heuristic estimate for ordering is based on how many of the input and output behavioral states match, and which state features and how many of them match in the target functional specification and the function of the candidate design analogue. In the situation where both a low-acidity NAC and a low-acidity Sulfuric Acid cooler were selected, IDEAL would order the latter as a better match than the former because the Sulfuric Acid cooler matches with the desired function on the *substance* also.

Although the above description of the selection and ordering tasks focused on probing the memory of analogues with a desired functional specification, those methods for matching on structural constraints of the given problem are very similar. Since the structural organization of analogues in memory is also similarly hierarchical, the same kind of search down the hierarchy as in Figure 4.5 works except that the matching now is on structural relations, components and substances, and their properties, as opposed to input and output states of a desired function. Similarly, the ordering of analogues is based on structural differences as suggested by the SBF language, and the heuristic estimate of the degree of match is based on how many of the structural relations, and component-parameters and substance-properties in the structural constraints of the target design problem match with those of the candidate design analogue.

4.3 Storage of the Target Design Analogues

When a new problem is solved and the target design solution is generated (or is acquired from an oracle in case of problem-solving failure), it needs to be stored in analogue memory for future use. Thus the final task in solving problems by analogy is to store the target analogues. For instance, IDEAL stores the new designs it has generated in its analogue memory for later use. In order for a new design analogue to be useful in later problem solving, it needs to be stored in “right” place. That is, the new design needs to be appropriately indexed by its functions and structural constraints because device-design problems specify both the desired functions and the structural constraints that the desired designs should satisfy. Since the analogue memory may not have all the indices predetermined, storing a new analogue requires the indices to be determined for the analogue. If the analogue memory is hierarchically organized, then learning of new indices and even the addition of the new analogue causes the need for re-organizing the memory; it is necessary in order for the future retrievals to be efficient and effective. Since in the representation of analogues in MBA the design analogue points to the SBF model of the

design, the newly learned case-specific SBF model of the device is also stored in the memory. (Adaptation of design analogues and learning of SBF models will be discussed in later chapters.)

4.3.1 Learning of Indices to Target Design Analogues

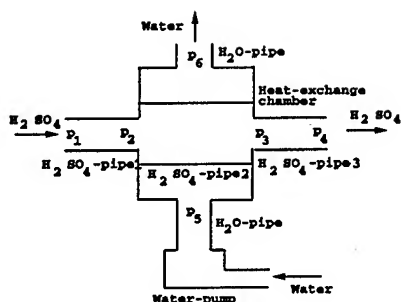
Generalization-Specialization hierarchies provide a very general model for organizing the analogue memory. Since analogues can be discriminated along multiple dimensions, the analogue memory in general may contain multiple generalization-specialization hierarchies. In order to be able to retrieve an analogue when a new problem is specified, the vocabulary used for indexing the analogues has to match the dimensions of generalization/specialization. In addition, the level of generalization at which an analogue is stored also determines whether it will be retrieved when a new problem is given. Storing an analogue in such an analogue memory thus implies two distinct issues in index learning: *learning the indexing vocabulary* and *learning the right level of generalization*. Deciding on the indexing vocabulary generally requires some notion of what is important about the new analogue and the task for which it is likely to be reused. The level of generalization depends in part on the analogues already stored in memory and the inductive biases that can be generated at storage time.

In model-based analogy, we explored the hypothesis that the SBF model of a design, together with a specification of the task for which the design analogue may be reused, provides the vocabulary for indexing the design analogue in memory. Furthermore, we explored how the model-based method, together with similarity-based learning (using earlier design analogues in memory) helps to determine the level of index generalization.

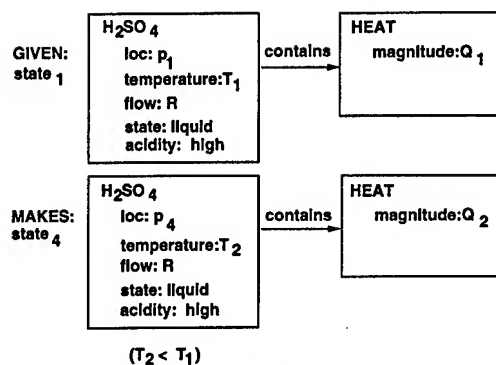
4.3.1.1 Functional Indices

Recall that the design analogues in IDEAL are indexed by their functions and are organized in generalization-specialization hierarchies along the dimensions of properties of substances. In order to illustrate IDEAL's learning of indices, let us suppose that its memory has a snapshot as shown in Figure 4.9(a).

Now consider the task of identifying indices for the design of high-acidity SAC (Figure 4.7(a)), whose structure, function, and behavior are shown in Figures 4.7(b), 4.7(c), and 4.7(d) respectively, before storing it in memory. This design may have been generated by adapting the similar design analogue, for instance the design of low-acidity NAC, currently available in memory (Goel, 1991a) or it may have been given directly by an oracle. Although the analogue memory presently has designs of acid coolers organized only along the dimension of property *acidity*, the new design analogue may better be indexed along other dimensions also so that it is more useful in future design episodes. In general, there are two different issues concerning the selection of functional indices for the new design analogue. First, if a new design is stored only along the substance properties specified in its function, the retriever would not be able to make use of knowledge of other substance properties relevant to the design. Second, if the new



(a) High-Acidity Sulphuric Acid Cooler



BY-BEHAVIOR : pointer to the behavior "Cool Acid"

(c) Function "Cool Acid" of High-Acidity SAC

STRUCTURE High-Acidity SAC

COMPONENTS: (Heat-Exchange-Chamber
H₂SO₄-pipe-1
H₂SO₄-pipe-2
Water-pump
...)

STRUCTURE Heat-Exchange-Chamber

RELATIONS: (SERIALLY-CONNECTED
Heat-Exchange-Chamber
H₂SO₄-pipe-1)
(INCLUDES
Heat-Exchange-Chamber
H₂SO₄-pipe-2)
...

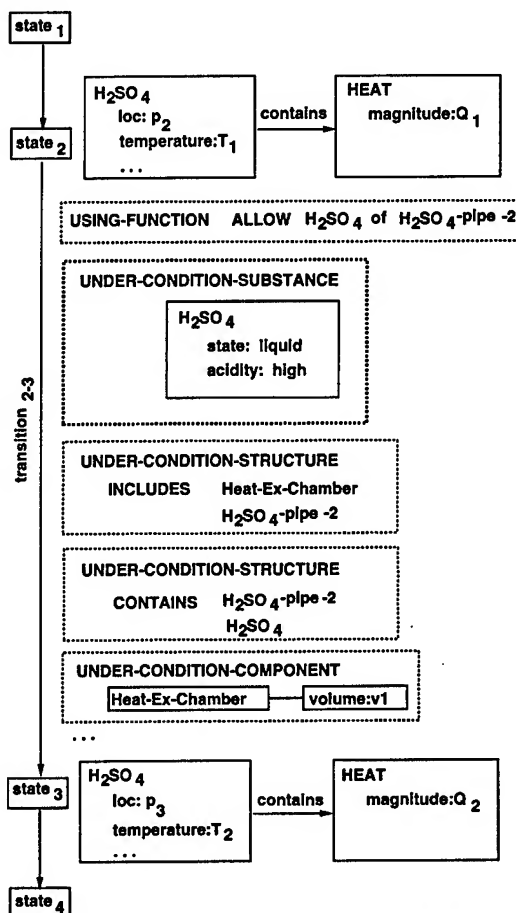
PARAMETERS: (volume v1)

FUNCTIONS: (ALLOW Water)

CONNECTING-POINTS: (p2 p3 p5 p6)

...

(b) Structure of High-Acidity SAC in Schema Form



(d) Behavior "Cool Acid" of High-Acidity SAC

Note: All locations are with reference to components in this design.
All labels for states and transitions are local to this design.

Figure 4.7: Design of High-Acidity Sulfuric Acid Cooler

design is indexed by all the properties of the substance in its functional specification, then the analogue retriever may retrieve a design based on a match with an unimportant property, which can make adaptation hard or even impossible. Hence, the issue becomes how to determine the substance properties that are *relevant* to the functioning of the design. Therefore, the important issue is how to learn *new* indexing vocabulary.³

4.3.1.2 Model-Based Learning of Functional Indices

In MBA, SBF models provide the knowledge to constrain these tasks of index learning. The process of MBA capitalizes on the knowledge of the causal behavior in the SBF model of the new analogue. In particular, it uses the behavioral requirements on the substance expressed under UNDER-CONDITION-SUBSTANCE to identify the substance properties relevant to the functioning of the design. These behavioral requirements of a substance specify that in order for the transition to take place the properties of the specified substance should satisfy certain conditions and hence are important to the design. Thus they help to learn new indices appropriate for the new design.

IDEAL's algorithm for selecting useful indices to a new analogue is shown in Figure 4.8. Given a new design analogue and the type of indexing (i.e., functions), this method traverses through the causal behaviors in the SBF model of the design to identify substance properties on which the working of the design is predicated. Since the SBF model can specify multiple behaviors, the outer loop in the algorithm analyzes each causal behavior in the model. The second loop is for analyzing the transitions of a causal behavior. If a substance property is a part of the causal context of a transition,⁴ then the algorithm adds it to the set of indexing features if it is a property of the containing substance in the functional specification; and, it adds it to the set of alternative indexing features if it is also in the parameter-relations on the transition. Since the causal behaviors in IDEAL's SBF model are specified at different levels of detail, the algorithm searches the space of behaviors in a breadth-first manner. If a higher level behavior does not lead to the identification of any useful substance properties, then the more detailed behavior, indicated by *by-behavior*, is added to the list of plausible sources of indexing features.

For example, given the functional specification of high-acidity SAC (Figure 4.7(c)) and its causal behavior (Figure 4.7(d)), the above method results in *acidity* and *state* as the indexing features for storing this analogue in memory. This is because the annotation on *transition₂₋₃* specifies that the transition can occur only under certain conditions on properties *state* and *acidity* of the substance flowing through *H₂SO₄-pipe-2*. The initial analogue memory (Figure 4.9(a)) did not have the property *state* as part of its indexing vocabulary. The SBF model

³By *new* indexing vocabulary, we do not mean that the vocabulary is new to IDEAL but rather it is new for the purpose of indexing.

⁴*get-under-conditions* in the algorithm gets the annotations such as UNDER-CONDITION-SUBSTANCE and UNDER-CONDITION-COMPONENT from the given transition corresponding to the type of indexing used.

Input:

- Design analogue, **C**, that needs to be stored.
- Functional specification of the design, **F**.
- Type of indexing, **T**, that is, functional.
- One causal behavior (subset of model), **M**, corresponding to **F**.

Output: Exact vocabulary for indexing **C**, i.e., the set of useful features from **F**.

Procedure :**initialize**

containing-substance-props **P** = get-containing-substance-properties(**F**);
 indices = alternative-indices = plausible-sources-of-indices = {};

while true do

1. **foreach** causal behavior **B** \in **M** **do**

• **foreach** transition **t** \in **B** **do**

- conditions-on-features = get-under-conditions(**T**, **t**);
- indices = indices \cup {**f** | feature **f** \in conditions-on-features \wedge **f** \in **P**};
- alternative-indices = alternative-indices \cup {**f** | feature **f** \in conditions-on-features \wedge **f** \in **P** \wedge parameter-relations(**t**)};
- **if** indices = **P** **then** exit(indices);
- **if** conditions-on-features = {} **then** plausible-sources-of-indices = plausible-sources-of-indices \cup get-detailed-behavior(**t**);

end

end

2. **if** plausible-sources-of-indices = {} **then**

- **if** indices \neq {} **then** exit(indices);
- **if** alternative-indices \neq {} **then** exit(alternative-indices);
- indices = {**p** | **p** \in **P** \wedge input-state-value(**p**) \neq output-state-value(**p**)};
if indices \neq {} **then** exit(indices);
- indices = {**p** | **p** \in get-contained-substance-properties(**F**) \wedge input-state-value(**p**) \neq output-state-value(**p**)};
if indices \neq {} **then** exit(indices);
- exit(**P**);

3. **M** = plausible-sources-of-indices;

4. plausible-sources-of-indices = {};

end

Figure 4.8: A model-based method to obtain functional indices for design analogues

however suggests that **state** is a useful index to the new design analogue, and so IDEAL indexes the new analogue by **state** also. A snapshot of the analogue memory after storing this design is shown in Figure 4.9(b).

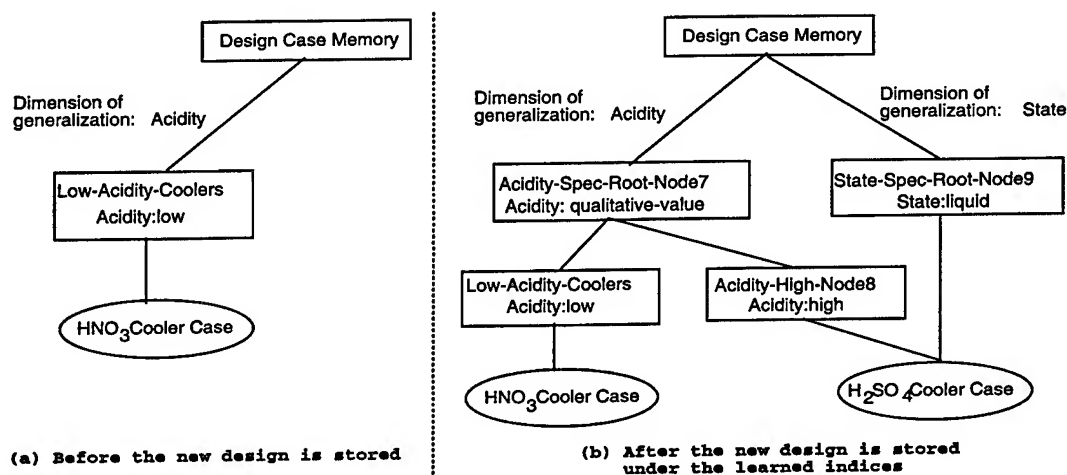


Figure 4.9: Snapshots of IDEAL's functionally organized analogue memory

Once the indexing features are selected, IDEAL uses similarity-based learning to generalize them. Under each property, IDEAL organizes the analogues in a binary tree discriminated on values of the property in the analogues. It uses the differences in the values of a given property that constitute a type of functional difference between two designs to determine whether the two designs belong to the same category or to different categories. For example, the design of high-acidity SAC is stored under the category of **Acidity-High-Node8** that is different from that of **Low-Acidity-Coolers** (Figure 4.9(b)) because their values of acidity differ. The level to which the indices are generalized depends on how similar are the corresponding values in the new and old analogues in memory. For instance, a more general category **Acidity-Spec-Root-Node7** is created that covers both low and high values of acidity.⁵ Note that H_2SO_4 Cooler Case is stored in multiple levels corresponding to the nodes **Acidity-Spec-Root-Node7** & **Acidity-High-Node8** under the property acidity and at one level corresponding to the node **State-Spec-Root-Node9** under the property state.

4.3.1.3 Structural Indices

IDEAL uses the structural relations between different components in a design structure and the parameters of components in the design as dimensions for organizing design analogues struc-

⁵The general value of acidity at this higher-level node comes from IDEAL's knowledge of qualitative values and quantitative values. If the values are qualitative, it is determined by climbing up a known value hierarchy. And, if the values are quantitative, a new value range is created that spans from the lowest of the two child nodes in the discrimination tree to the highest of the two.

turally. That is because some class of design problems involve specifications of structural relations desired and values or ranges desired for parameters of components to be used in the designs. For example, as shown in Figure 4.3(a), it organizes *acid coolers* along the dimension of structural relation (INCLUDES Heat-Exchange-Chamber HNO_3 -pipe-2) at one level, and further discriminates between analogues based on values of component parameters. The HNO_3 cooler case that cools HNO_3 from T_1 to T_2 is stored under the structural relation (INCLUDES Heat-Exchange-Chamber HNO_3 -pipe-2) and along the parameter capacity of the pipe in Figure 4.3(a).

4.3.1.4 Model-Based Learning of Structural Indices

Now consider the task of identifying structural features as indices for the design of high-acidity SAC (Figure 4.7). This task is analogous to, but different from, that of learning functional indices. The same issues as we have seen in learning of functional indices arise in this task as well. Again, IDEAL capitalizes on the knowledge of the causal behavior in the SBF models in order to address these issues. It uses the behavioral and structural requirements on the structure expressed under UNDER-CONDITION-COMPONENT and UNDER-CONDITION-STRUCTURE respectively to identify the structural features relevant to the functioning of the design.

IDEAL's algorithm for selecting useful structural indices to a new analogue is shown in Figure 4.10. Given a new design analogue and the type of indexing (i.e., structural features), this method traverses through the causal behaviors in the SBF model of the design to identify structural relations and component parameters on which the working of the design is predicated. Note that the method in Figure 4.10 is very similar to that in Figure 4.8—the major difference is in the types of context captured in the transitions of SBF models that come to bear upon the task; context stored in UNDER-CONDITION-SUBSTANCE is used for learning functional indices while the contexts in UNDER-CONDITION-COMPONENT and UNDER-CONDITION-STRUCTURE are used for learning structural indices.

For example, given the structure of high-acidity SAC (Figure 4.7(b)) and its causal behavior (Figure 4.7(d)), this method results in the structural relations (INCLUDES Heat-Exchange-Chamber H_2SO_4 -pipe-2) and (CONTAINS H_2SO_4 -pipe-2 H_2SO_4), and the component parameters capacity of H_2SO_4 -pipe-2 and volume of Heat-Exchange-Chamber as structural indices for storing the design of high-acidity SAC. This is because the annotation on transition23 specifies that the transition can occur only under certain structural relations between components and certain conditions on parameters of components. Note that the structural organization of analogue memory after storing this design (Figure 4.3(b)) has new indices, namely, the structural relation (CONTAINS H_2SO_4 -pipe-2 H_2SO_4) and the parameter volume of Heat-Exchange-Chamber.

Input:

- Design analogue, **C**, that needs to be stored.
- Structural description of the design, **S**.
- Type of indexing, **T**, that is, structural.
- All causal behaviors (model), **M**, including one for the function.

Output: Exact vocabulary for indexing **C**, the set of useful features from **S**.

Procedure :**initialize**

```

component-parameters P = get-component-parameters(S);
component-structural-relations R = get-component-structural-relations(S);
indices-strl-relations = indices-comp-params = {};
plausible-sources-of-indices = {};

```

while true do1. **foreach** causal behavior **B** \in **M** **do**• **foreach** transition **t** \in **B** **do**

- indices-strl-relations = indices-strl-relations
 \cup get-under-conditions(**T**, **t**);
- indices-comp-params = indices-comp-params \cup {(**p**, **c**) | **p** is a parameter of component **c** \wedge
 $\mathbf{p} \in (\text{parameter-relations}(\mathbf{t}) \cup \text{get-under-conditions}(\mathbf{T}, \mathbf{t}))$ };
- **if** ((indices-strl-relations = **R**) \wedge (indices-comp-params = **P**)) **then**
 exit(indices-strl-relations, indices-comp-params);
- **if** ((indices-strl-relations = {}) \wedge (indices-comp-params = {})) **then**
 plausible-sources-of-indices =
 plausible-sources-of-indices \cup get-detailed-behavior(**t**);

end

end

2. **if** plausible-sources-of-indices = {} **then**

- **if** ((indices-strl-relations \neq {}) \wedge (indices-comp-params \neq {})) **then**
 exit(indices-strl-relations, indices-comp-params);
- **if** indices-strl-relations = {} **then** indices-strl-relations = **R**;
- **if** indices-comp-params = {} **then** indices-comp-params = **P**;
- exit(indices-strl-relations, indices-comp-params);

3. **M** = plausible-sources-of-indices;

4. plausible-sources-of-indices = {};

end

Figure 4.10: A model-based method to obtain structural indices for design analogues

CHAPTER V

NON-LOCAL MODIFICATIONS AND CROSS-DOMAIN TRANSFER: USE OF GTMS

In the previous chapter, we described the memory issues in the MBA process such as indexing and organization of analogues, retrieval of analogues, index learning, and storage. In this chapter, we discuss the issues of non-local modifications and cross-domain transfer that arise in the transfer and modification subtask of the MBA process. This subtask is to transfer the solution in the retrieved source analogue to the target problem and modify the solution to fit the target problem. Retrieval of a source analogue can lead to one of several conditions: no match, exact match, and partial match. If no analogue is retrieved for the target problem, the MBA process fails to solve the problem. If the match between the target problem and the retrieved source analogue were exact, then the transfer of the solution from the source analogue to the target problem is trivial because no aspect of the solution needs to be modified. On the other hand, adaptation (i.e., transfer and modification) becomes necessary in the MBA process when the source analogue is a partial match to the given problem. This task raises two issues.

The first issue is to identify "what" in the source design analogue needs to be modified, given the differences in the source problem and the target problem. That is, the task is to set up adaptation goals. Differences between the source and the target problems are computed in the retrieval stage of the MBA process. In some domains such as the design of physical devices, since a single difference in the problems can map onto multiple differences in the solutions (i.e., a *one-to-many* relationship), the reasoning required to solve this task can be very complex. The reasoning can become very expensive especially when there are multiple differences in the source and target problems. It is even harder in analogical reasoning because the target solution is not available yet at this stage and the modifications to the source design solution need to be identified. Therefore, the issue becomes how to control this reasoning. In MBA, our solution is to use the knowledge of device models to guide the process. The SBF representations of device models are especially useful because they explicitly capture how the functions of structural elements get composed into the overall device function and what aspects of the device are important for its functioning. That is, the knowledge of composition encapsulated in the device model helps in decomposing the device and thus controlling the inferences in identifying structural modifications for the given functional differences.

The second issue is how to actually make the modification to the solution in the source design analogue given a difference in the problems. That is, the task is how to achieve adaptation goals.

To make the modification computationally tractable, it is desirable of a computational process to localize the modification to a small structural element when possible. In device design, once a structural element is localized for modification for a given functional difference, there can be multiple ways of realizing the modification to reduce the difference. The selection of a strategy depends on the specific functional difference, the structural elements available, and the very knowledge of making modifications (i.e., the strategic knowledge). A simple modification such as replacing a selected structural element with a new one is not always sufficient and may not even be possible due to the above dependency. Moreover, it may not always be possible to localize the functional difference to a modification to a structural element. Therefore, there is a need for multiple strategies for handling all these different types of modifications (for instance, local and non-local). In the MBA process, we use different strategies for achieving different adaptation goals. One of the central foci in this work is on making non-local modifications. Making non-local modifications can be hard because of the reasoning required to infer interactions between non-local elements and take care of their effects. Therefore, to control the reasoning, there is a need for design knowledge that encapsulates the relationships and dependencies between non-local modifications and their effects. In MBA, our solution to this issue is to use design patterns. In this chapter, we describe how GTMs are used for this purpose in the MBA process. But first, let us indicate how the MBA process integrates two different types of transfer.

Multiple Types of Transfer: Depending on the level of abstraction of the retrieved knowledge (i.e., a source design analogue or a design pattern) and the source of strategic knowledge for adaptation, there are two different types of transfer mechanism possible in the MBA process: (1) modification of the retrieved design to achieve the functionality of the desired design and (2) transfer of design knowledge from a different design domain. The first type of transfer assumes that an adaptation strategy that makes the needed modification is available in the domain of the target problem. This type of transfer is possible when the retrieved knowledge is a specific design from the same domain as the target problem. We call the process involved in this type of transfer model-based adaptation. The second type of transfer is possible when the retrieved knowledge is an abstraction such as a design pattern learned in a different domain.

In this chapter, we will describe the two types of transfer processes at two levels in a nested manner: at the outer level, we describe the process of model-based adaptation between specific designs (i.e., transfer process-1); and at the inner level, we describe the process of instantiation of a design pattern such as GTM (i.e., transfer process-2) to solve a subtask of transfer process-1. Thus transfer process-2 supports transfer process-1 in MBA. Figure 5.1 illustrates this relationship between the two types of transfer processes. The transfer of design patterns occurs for achieving the adaptation goals that arise in transferring between specific designs. This will become clearer as we walk through the process(es) in the following sections.

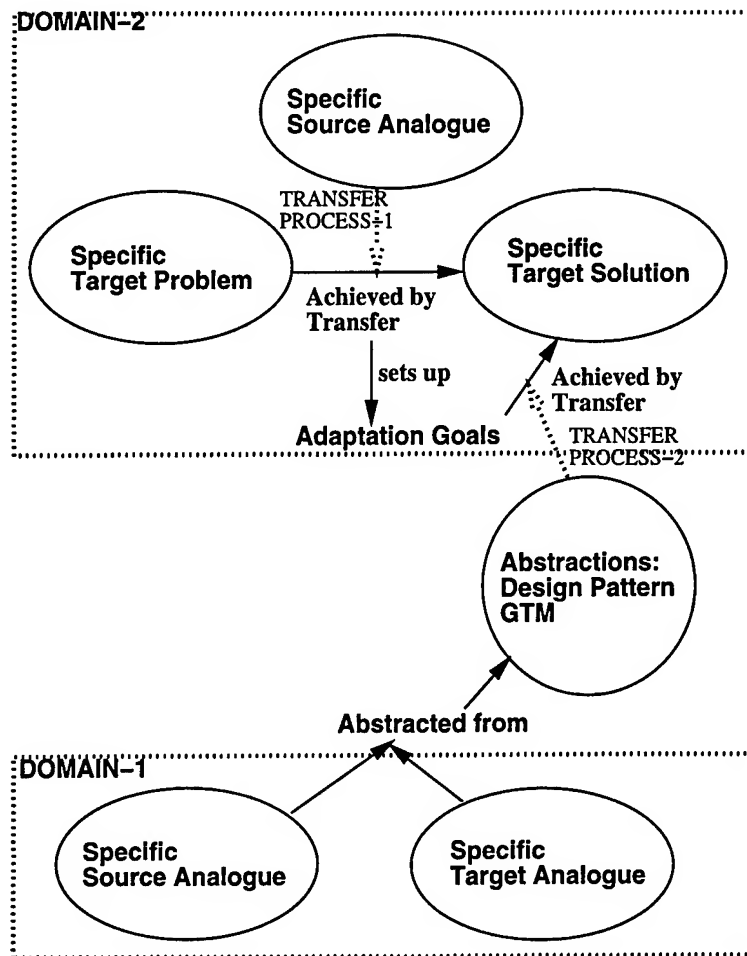


Figure 5.1: Illustration of The Relationship Between Two Types of Transfer Processes in IDEAL

5.1 Spawning of Adaptation Goals

When there are differences between the functions specified in the target problem and the source analogue, IDEAL sets up the task of spawning the adaptation goals—the specific goals for adapting the partial design solution in the source analogue. (Computation of functional differences was described in the previous chapter in the context of ordering retrieved analogues.) This task involves localizing (if possible) the possible “faults” in the source design that when fixed can help to deliver a solution to the target problem, given the functional differences to be reduced. In other words, this task is to propose candidates for modification that can help to deliver the desired function. That is, this task of diagnosis takes as input the desired function, a source design analogue, and the functional differences between the source and the desired, and gives as output a set of possible faults. Each possible fault can be described as a triple consisting of either a substance or a component in the source design, a property of that substance or compo-

ment, and a relation between that property and some property in the output state of the desired function. In order to control the complexity of reasoning in this task, we use the knowledge of device models in the process for MBA. That is, in the model-based approach, the device model indexed off the source analogue guides the localization and identification of the faulty parts that need to be repaired.¹

The task of mapping a given set of functional differences to the candidate structural modifications is in general very hard, especially when there are multiple differences and the needed modifications are non-local, many, and interacting. In order to deal with the computational complexity of this task, IDEAL uses the knowledge of device models to constrain the process. That is, it uses the algorithm shown in Figure 5.2, which given the functional differences and the SBF model of the source design traces through the model and identifies specific structural elements (components or substances) that need to be modified.² The functional differences to be reduced and a selected modification together constitute an adaptation goal.

If the function desired of a design and the function delivered by the device in the source design analogue differ in more than one feature, then IDEAL heuristically ranks the differences in order of the difficulty of reducing them. For instance, in the domain of physical devices that can be modeled in terms of the flow of substances between components, reducing the difference *substance1* \rightarrow *substance2* is heuristically believed to be less difficult than reducing *value1* \rightarrow *value2* of a property.

Let us now trace through the algorithm given in Figure 5.2 in order to understand IDEAL's process of identifying possible faults in the source design. IDEAL first accesses the SBF model of the source design using the function of that design, that is, using the pointer *by-behavior* in the function schema. Then it traces the behavior backwards starting from the final state and identifies the substance properties or component parameters that are related to the property in the output state of the candidate design function. In the traversal, it finds the transitive closure of the property that needs to change, that is, all the properties that are either directly related to this property or that are related to this property through one or more properties.³ In order to avoid infinite traversals through the causal behaviors with loops, IDEAL keeps track of all the visited states and checks if it has already visited the current state; and if the answer is 'yes' then it stops.

The loop in the BACKTRACE procedure essentially considers if the knowledge encapsulated

¹IDEAL's method for diagnosis basically comes from its predecessor system KRITIK2. The original algorithm has been reported in (Stroulia et al., 1992). In this work, we have adapted it to apply for diagnosing "non-linear" causal behaviors that involve causal loops, and forks and joins in a causal sequence.

²In this and the subsequent algorithms, the notation *slot-name(schema)* denotes the value of a slot in a schema. For example, *PropertyList(State_{OUT}(F))* denotes the value of the slot *PropertyList* in the schema for *State_{IN}* which in itself is a slot of the schema for function *F*. All these notations are similar to those introduced in the formal descriptions of SBF models in Chapter 3.

³Suppose that P is the property to change. If there is a relationship between P and another substance property or component parameter Q, then Q is added to the set. Similarly, if there is a relationship between Q and another property R, then R is also recursively added to the set.

Input: • Desired function, F_{new} .
 • Source design analogue, C .
 • Functional difference, $FD = F_{new} - F_{old}$, where F_{old} is the function in C .

Output: • A set of possible faults (or candidate modifications), $S_{possible-faults}$.

Procedure:

initialize

$F_{old} = \text{functional-spec}(C)$;
 $S_{possible-faults} = \{(E, P_i, \text{Relation}) \text{ where}$
 $E \in \{\text{Sub}, \text{Comp}\}, P_i \in \text{PropertyList}(E)$, and
 $\exists P' \in \text{PropertyList}(\text{State}_{OUT}(F_{new})) \text{ s.t. } P' \text{Relation } P_i$ holds good
 and P' is the property whose value needs to be changed,
 and $\text{Relation} \in \{\text{directly-proportional-to}, \text{inversely-proportional-to}\}\}$;
 $B_{old} = \text{function-by-behavior}(F_{old})$; visited-states = $\{\}$;

begin

$S_{possible-faults} = \text{BACKTRACE}(\text{behavior-final-state}(B_{old}), S_{possible-faults}, \text{visited-states})$;

end.

BACKTRACE (state, $S_{possible-faults}$, visited-states)

current-state = state;

LOOP: previous-state = state-previous-state (current-state);

if previous-state = NIL \vee current-state \in visited-states, **then exit LOOP**;

case :

(1) **if** there is a qualitative equation in state-previous-transition(current-state),
 and property $P_i \in S_{possible-faults}$ such that $P_i^{old} \text{Relation } P_i$, where
 P_i is a parameter of a component or a property of another substance E ,
 and P_i^{old} is a property in the current design,
 then $S_{possible-faults} = S_{possible-faults} \cup \{(E, P_i, \text{Relation})\}$
 where $P_{i, \text{value}}^{new} \text{Relation}^{-1}(P_{i, \text{value}}^{new})$, and
 $P_{i, \text{value}}^{new}$ is the value of P_i in the desired design;

(2) **if** there is a functional abstraction of a component E (i.e., Comp in
 USING-FUNCTION) and a condition on the substance properties on which
 E operates (i.e., Condition_{SUB}) in state-previous-transition(current-state),
 and property $P_i \in S_{possible-faults}$ such that $P_i^{old} = P_i$, where
 P_i is a property of the substance in Condition_{SUB} , and
 P_i^{old} is a property in the current design,
 then $S_{possible-faults} = S_{possible-faults} \cup \{(E, P_i, \text{directly-proportional-to})\}$
 where $P_{i, \text{value}}^{new}$ determinesthetypeofcomponent E , and
 $P_{i, \text{value}}^{new}$ is the value of P_i in the desired design;

(3) **if** there is a pointer to a new behavior sequence B' such that the transition
 state-previous-transition(current-state) depends on a transition, tr , of B'
 then **BACKTRACE** (transition-next-state(tr), $S_{possible-faults}$, visited-states);

end-case

visited-states = visited-states \cup {current-state};

current-state = previous-state; **goto LOOP**;

end.

Figure 5.2: IDEAL's Method for Spawning Adaptation Goals

in the previous transition of the current state indicates whether any of the current possible faults are related to some substance property or component parameter; if the transition does indicate so, then the particular property is added to the set. There are three different types of knowledge in the transition that may lead to adding a new property to the set of possible faults: (1) if there is a qualitative equation in the transition that relates a property in the current set of possible faults to a substance property or a component parameter, then that property is added to the set (i.e., actually, the triple consisting of substance/component, its property/parameter that is related to one in the current set, and the relationship between the properties/parameters); (2) if a component contributes a function to the transition, and there is a condition on the property of a substance on which the component operates, and if that property is related in an equality relationship to a property in the current set of possible faults, then that property is added to the set (the relationship between the properties is treated as *directly-proportional-to*); and (3) if the transition is dependent on a transition in another internal behavior of the device, then that behavior is also traversed in the same way as the current one, starting from the next state of the transition in that behavior.

At the end of the process described above, IDEAL finds all the possible candidates (substances and components) for modification. But, it is not necessary that for all types of functional differences IDEAL would find the set that is a proper subset of the substances and components in the entire structure of the candidate design (that is, it may not be able to localize the modifications to a subset of the structural elements.). For instance, when the functional difference is of the type substance-property-value-fluctuation difference, IDEAL cannot localize the modification because such a difference is not localizable. We will present an example with such a functional difference later in this chapter.

When there are multiple candidates for modification to reduce the functional differences, IDEAL tries them one-by-one until either it is able to successfully modify the candidate design to deliver the desired function or it exhausts all the possible modifications (and it fails). When none of the modifications to the current candidate design lead to a solution, then IDEAL tries to adapt a different source design, if there is one; otherwise, it fails.

5.2 Multiple Adaptation Strategies

Once a set of possible candidate modifications is identified for the candidate design, the task is to choose one and execute it in the candidate design. In general, different types of modifications require different kinds of adaptation knowledge. Therefore, to facilitate selection of an appropriate strategy for a given modification, we need a typology of modifications.

A Typology of Structure Modifications: The SBF language provides a vocabulary for expressing certain types of modifications to the structure of a design. The typology of structure modifications (or, in other words, adaptations) includes the categories of substance substitution (including substance generalization and specialization), component replacement, substance-

property modification, component-parameter modification, and component (or in general, device substructure) composition (i.e., component addition), and instantiation of a GTM (including structure replication or cascading, feedback control, and feedforward control mechanisms). In this work, the central focus is on the adaptation strategy that uses the knowledge of GTMs.

Recall that in IDEAL the functional differences to be reduced and a selected modification together constitute an adaptation goal. The adaptation goals are used as probes into the memory to retrieve the applicable adaptation strategies (otherwise also known as repair strategies). The task of "repair" is to execute a candidate modification given a set of possible modifications and strategies. In the context of design, it involves modifying the candidate design so that the possible faults are eliminated. That is, it takes as input the desired function, the candidate design analogue, functional difference between the desired and the candidate, and a set of possible faults, and gives as output a modified model and the corresponding function.⁴ It uses two types of knowledge: knowledge of functional differences and knowledge of adaptation strategies. In model-based analogy, the application of an adaptation strategy includes both repair and evaluation of the candidate solution. (Hence, these strategies may also be referred to as repair & evaluation strategies.) In IDEAL, these two steps are interleaved because that helps in avoiding the modification of the device structure in case it would not achieve the desired function.

Given a functional difference, there may be more than one potential structural modification possible, any of which when applied can result in reducing the difference. And, similarly, given a selected modification, there may be multiple strategies applicable to achieve that modification. When such multiple strategies are applicable, IDEAL chooses the simpler ones (i.e., those that make **local modifications**, for instance, the strategies of substance substitution and component replacement) prior to the more complex ones (i.e., those that make **non-local modifications**, for instance, the strategies corresponding to the instantiation of different GTMs). Furthermore, when a functional difference involves differences in multiple constituents (for instance, in multiple properties of the input substance or output substance), reducing the functional difference may require application of several strategies one after another.

We will now briefly describe each of the above modifications and the corresponding strategies in IDEAL, and then later focus on the strategy of instantiation of GTMs. For details on the types of modifications, see (Goel, 1989).

Substance Substitution: In this type of structure modification, a substance in the candidate design analogue is *substituted* by another substance. This is one of the simplest modifications in design, and it does not result in any changes to the structural topology of the candidate designs.

Substance-Property Modification: This type of modification is applicable when the functional difference is of the type substance-property-value difference. That is, when the desired function differs from the function in a candidate design in the value of a substance property. Making this modification may require modifying a component that affects the substance prop-

⁴The structure of the candidate design analogue is modified only after evaluating the modified behavior by simulation.

erty. Such component modification may be simply a component-parameter modification or component replacement, or it may even involve component addition or instantiation of cascading GTM in the particular component. Thus even achieving substance-property modification may result in changes to the structural topology of a candidate design under the additional structural constraints on the available components.

Component Replacement: In this type of modification, a component in the candidate design analogue with a specific function is replaced by another component that achieves the new, different, localized function as determined by the desired function.

Component-Parameter Modification: This type of modification is applicable when the overall functional difference maps onto the difference between the value of a parameter of a component in the desired design and the corresponding value in the candidate design. For instance, a substance-property-value difference in the desired and candidate functions may map onto changing the parameter value of a component in the candidate design.

Component Addition (or Device Composition): This type of modification involves inserting (adding or composing) a component whose function is less than or equal to the local desired function in a candidate design. That is, when the component is (serially) added, the functional difference between the desired and the candidate designs reduces. In general, this modification may be repetitively applied to solve an adaptation problem. Further, the revision of the SBF model of the candidate design involves inserting the behavior segment that corresponds to the function of the added component after the state resulting as output of the component in the candidate design. Suppose that the problem is to adapt a simple electric circuit that produces light of intensity 6 lumens taking electricity of voltage 1.5 volts as input for generating a design to produce light of 12 lumens. That adaptation may involve the addition of a 1.5-volt battery when the battery is selected for modification. The component addition is a special case of the more general strategy of device composition. While component addition involves adding only a primitive structural element at a time, the more general device composition involves adding a device that may be composed of multiple structural elements to the candidate design.

Instantiation of GTMs: The instantiation of GTMs is applicable for achieving different types of structure modifications depending on the particular GTM instantiated and the particular functional difference to be reduced. For instance, this is applicable for reducing both the substance-property-value difference and the substance-property-value-fluctuation difference. While cascading GTM is applicable for reducing substance-property-value difference when there are structural constraints on the available components, the feedback GTM and feedforward GTM are applicable for reducing substance-property-value-fluctuation differences. More specifically, the feedback GTM is applicable when the fluctuation in the output substance property value of a device needs to be reduced, and the feedforward GTM is applicable when the fluctuation in the input substance property value needs to be reduced. We will describe the strategy of instantiating GTMs in much detail in the next section.

5.3 Adaptation by Instantiation of GTMs

GTM s can be used in the transfer process by their *instantiation* in the context of target problems and candidate designs. GTM s can also play other roles in analogical design: act as indices to different analogues and help in the retrieval process; and mediate the mapping between the source analogue and the target problem and help in the transfer process. In IDEAL's computational process, GTM s are learned from design analogues (as explained in the Chapter 7), stored in an abstraction memory, and used for solving new design problems by instantiation, where the original designs and the new problems might be from different device domains.

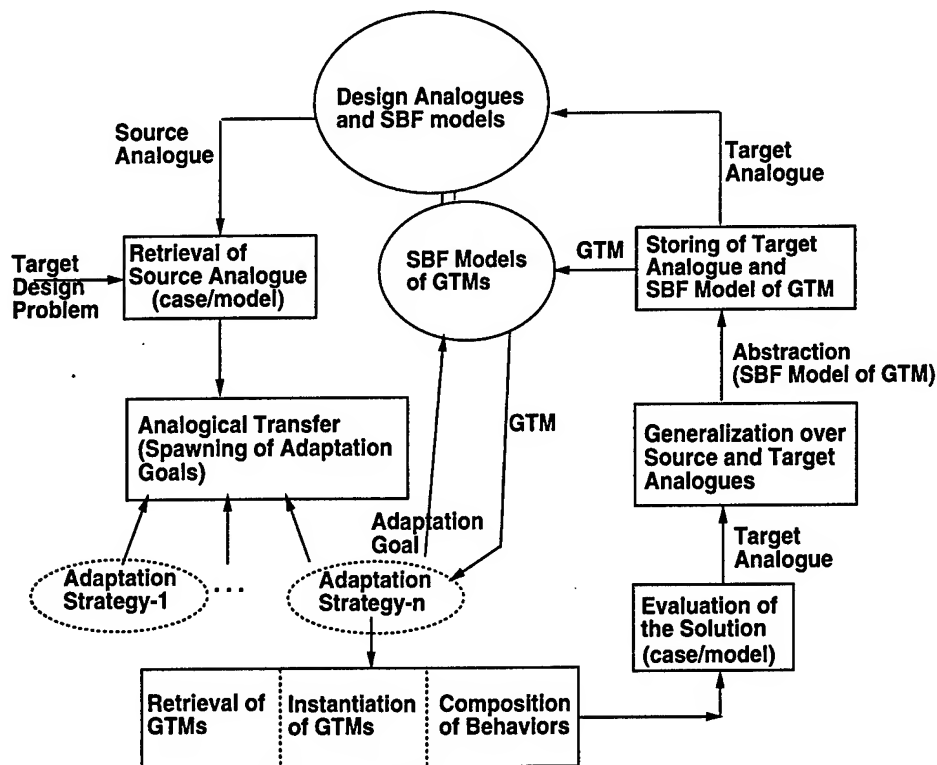


Figure 5.3: IDEAL's Partial Process of Analogical Design via GTMs

Figure 5.3 illustrates the IDEAL's partial process of analogical design via GTMs. (Compared to the process of model-based analogy in Figure 2.1, this figure excludes the stages that use GPPs and perform problem reformulation.) When a target design problem is presented, IDEAL first uses it as a probe into its analogue memory and retrieves a best matching source design analogue (and its SBF model). When there are differences between the functions specified in the target problem and the source analogue, it spawns the goal of adapting the design solution in the source analogue. Different types of functional differences lead to different types of adaptation goals,

some requiring only simple modifications (such as parameter tweaks) and some others requiring more complex modifications (such as topological changes). IDEAL first tries to perform simple modifications to the source design and considers more complex ones only when the simpler ones do not lead to a target solution. In order to control the reasoning involved in making complex, non-local modifications in design, a computational process such as in IDEAL needs knowledge that can encapsulate the relationships between non-local modifications and their effects; in device design, such relationships really are relationships between functional differences and the differences in causal behaviors, which exactly is the type of knowledge GTMs encapsulate. Therefore, IDEAL uses the knowledge of GTMs in making some types of complex modifications that involve changes to the device topology in the source analogue. It uses the adaptation goal as a probe into its memory of GTMs in order to retrieve a GTM and and instantiates the retrieved GTM in the context of the target problem. By instantiating a retrieved GTM, it first modifies the behaviors in the SBF model of the candidate design and then modifies the structure of the design.

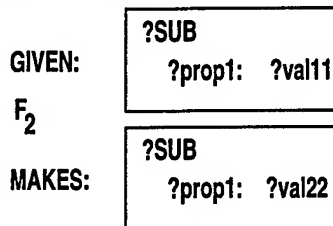
An Example GTM: Feedback Mechanism

Recall that in model-based analogy, the GTMs are also represented using the same SBF language as that used to represent device models. The SBF representation of a GTM encapsulates two types of knowledge: knowledge about the patterns of differences between the functions of known designs and desired designs that the GTM can help reduce; and knowledge about the patterns of modifications to the internal causal behaviors of the known designs that are necessary to reduce the differences. That is, it specifies relationships between patterns of functional differences and patterns of behavioral modifications to reduce those functional differences. For example, Figures 5.4(a) & (b) respectively show these two types of knowledge for a partial model of the feedback mechanism.⁵ Figure 5.4(a) shows the patterns of functions F_1 and F_2 respectively of a candidate design available and the desired design, and the conditions under which the mechanism is applicable. Because of the tasks for which they are used in model-based analogy, the GTMs are indexed by the patterned functional differences such as shown in Figure 5.4(a) (i.e., the fluctuations in the output substance property values are large vs. small). The model of the feedback mechanism indicates that the desired behavior (B_2) can be achieved by modifying the candidate behavior (B_1) through setting up the indicated causal relationships between the latter and the additional behaviors (that achieve the subfunctions of F_2 other than F_1 characterized in the applicability conditions of the mechanism). In particular, the feedback mechanism suggests the modification of looping back some output to the input and modifying the effective input to the device. Figure 5.4(b) shows (both diagrammatically and textually) the relation-

⁵Feedback can be open loop or closed loop in devices. Closed-loop feedback itself can be of 4 different types depending on the relationships between the output substance and feedback substance, and between the feedback substance and the input substance. The feedback mechanism described here is only one type of closed-loop feedback in which the output substance, feedback substance, and the input substance are all same. Also, there is no claim that there are only 4 types of feedback mechanisms; in general, there may be more.

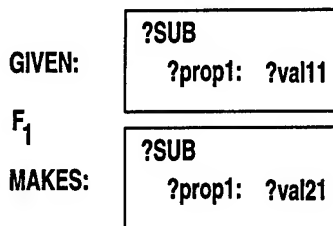
ships in a partial model of the feedback mechanism that IDEAL indeed learns from the designs of amplifiers (as explained in Chapter 7).

DESIRED DESIGN:



BY-BEHAVIOR: Behavior B2

CANDIDATE DESIGN:



BY-BEHAVIOR: Behavior B1

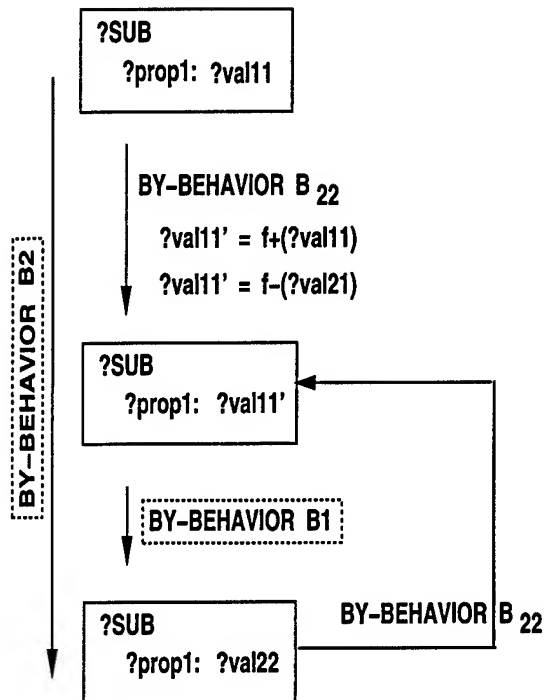
CONDITION:

$$?val22 \neq ?val21 ; ?val21 = ?val \pm \Delta$$

$$?val22 = ?val \pm \delta$$

$$F_2 = f : (?val11, ?val21) \rightarrow ?val11'$$

$$+ F_1 : (?val11) \rightarrow ?val21$$



$$B2 = B1 + B22$$

where B22 achieves function f

The relationships between B1 and B22 are such that:

$$\text{FINAL-STATE (B1)} \nsubseteq \text{INITIAL-STATES (B22)}$$

$$\text{FINAL-STATE (B22)} \nsubseteq \text{STATES (B1)}$$

Figure 5.4: The SBF Representation of A Feedback Mechanism

5.3.1 Retrieval of GTMs

We will now illustrate the IDEAL's use of GTMs in design by analogy. Consider a design problem it solves in the domain of mechanical controllers.⁶ Suppose that the new problem has a functional specification that given the substance angular momentum with a magnitude of L_i and

⁶The representations of the designs of gyroscope control system here are shown in more detail than their corresponding ones in Chapter 1.

clockwise direction at an input location (gyroscope), the device needs to produce the angular momentum with a magnitude L_o' proportional to the input and the same direction at a specified output location. In addition, it specifies the constraint that the output cannot fluctuate much around an average value (i.e., L_o' needs to be $L_{avg} \pm \delta$, where δ is a small fluctuation). This is the problem of designing a gyroscope follow-up (Hammond, 1958).

Let us also suppose that the design of a device (shown in Figure 5.5 (a), (b), & (c)) which transfers angular momentum from a gyroscope to an output shaft location is available in IDEAL's analogue memory (or is given explicitly as part of the *adaptive* design problem). The functional specification of the available device is that given an input angular momentum of magnitude L_i and clockwise direction at the input (gyroscope) location, it produces a proportional angular momentum of magnitude L_o and of clockwise direction at the output shaft location; however, L_o fluctuates over a large range, i.e., $L_o = L_{avg} \pm \Delta$, where Δ is large. Figure 5.5 shows the design of the available device: Figure 5.5(a) presents the functional specification of the device; Figure 5.5(b) shows the structure of the device schematically; and finally, Figure 5.5(c) shows the causal behavior of the device as a sequence of alternating states and state-transitions that explains the internal causal process of the device. IDEAL retrieves (if not given explicitly) the design of gyroscope control system available in analogue memory because the desired function matches with the function of this design.

Now, the task for IDEAL is to transfer and modify the candidate design of gyroscope control system to deliver the desired function. (Here, we are referring to the transfer process-1 illustrated in Figure 5.1.) Simple modifications such as replacing a component in the given design analogue will not result in a device that can solve the new design problem. It is because there is no single component in the device that seems responsible for the large fluctuations and that which may be selected for modification. Then the issue becomes if and how IDEAL can solve such a non-local adaptation problem using the knowledge of GTMs.

The first step for IDEAL in this process of analogical transfer is to retrieve the GTM. It uses the difference in the functions of the candidate and desired designs as a probe into its memory because it indexes the mechanisms by the functional differences and the decomposability conditions on the desired functions. It retrieves the feedback mechanism because the current functional difference, namely, the fluctuation in the output property is large vs. small (i.e., Δ vs. δ), matches with the difference that the feedback mechanism reduces which is specified in a device-independent manner. Then, it tries to match the decomposability condition on the desired function in the feedback mechanism (see Figure 5.4(a) for the condition $F_2 = \dots$) with the current desired function in order to find the subfunctions f (or g) that need to be designed for and composed with the candidate function. By performing this match, as guided by the language of SBF models, IDEAL finds the subfunction $f:(L_i, L_o') \rightarrow L_{ww}'$, that is, it needs to design for a structure that takes two inputs, angular momentum with a magnitude of L_i and angular momentum with a magnitude of L_o' , and gives as output an angular momentum of L_{ww}' .

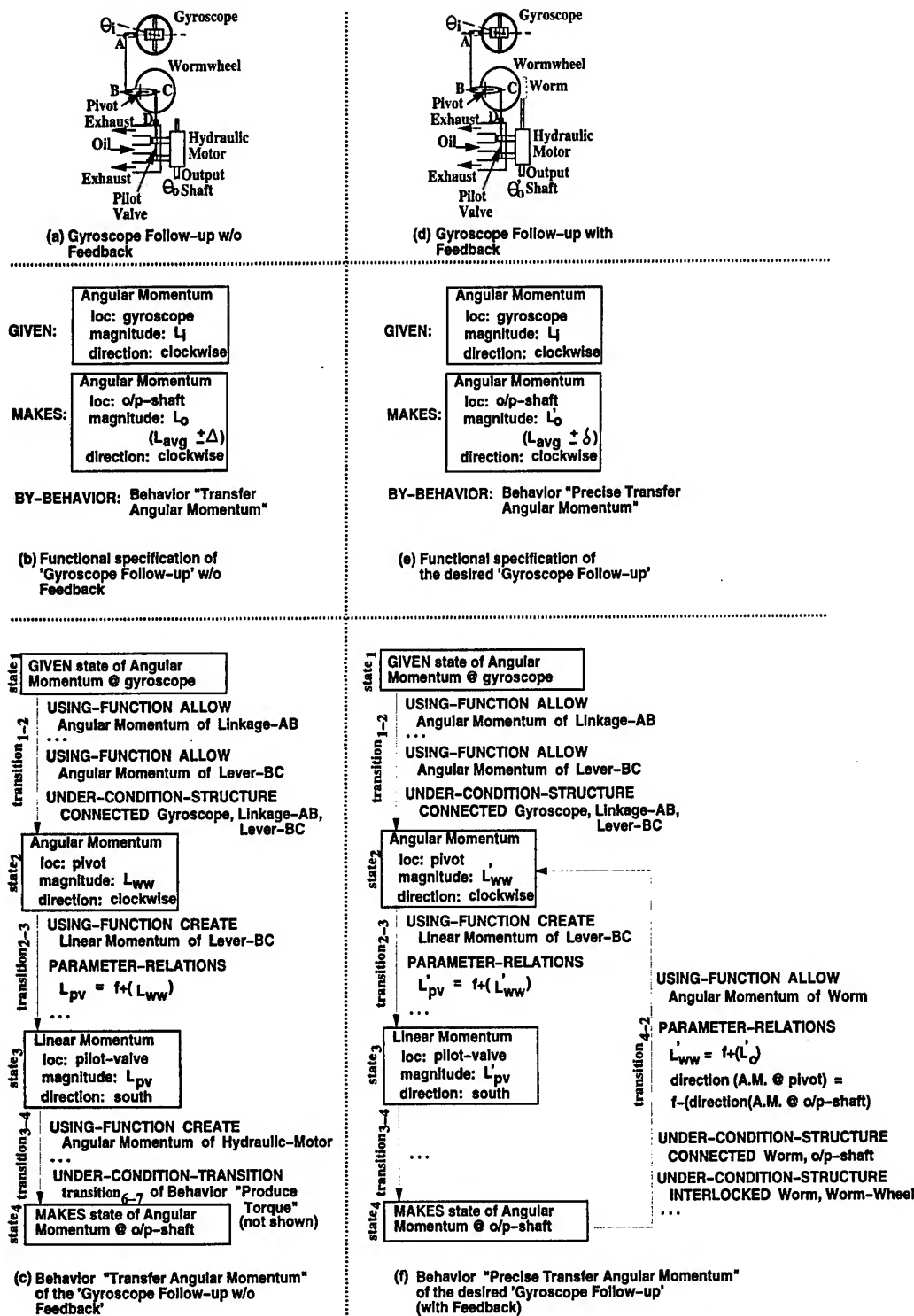


Figure 5.5: The Two Designs of Gyroscopic Follow-up *before* and *after* instantiating the Feedback Mechanism

in the opposite direction at the location of pivot in the candidate design.⁷

5.3.2 Instantiation of GTMs

The next step for IDEAL in this process is to transfer the retrieved GTM to the target design problem by instantiating it in the context of the target problem. (Now, we are referring to the transfer process-2 illustrated in Figure 5.1). The algorithm for IDEAL's process of instantiating the GTMs is shown in Figure 5.6. Some of the bindings for the state variables in the subfunctions of the GTM are obtained while doing the matching itself. But, some others need to be determined by tracing the relevant behavior of the candidate design backwards and identifying the possible states in the behavior that can be part of the subfunctions. For instance, in the example of gyroscope follow-up design, tracing back from *state*₄ in the behavior shown in Figure 5.5(c), IDEAL selects the *state*₂ as a possible candidate for instantiating the MAKES state of the subfunction *f*. *State*₃ is not a possible candidate because it specifies a different substance than what is already bound to the substance in the subfunction (i.e., linear momentum vs. angular momentum). Although in this example, there is only one candidate state for modification, in general, there can be multiple candidates. When there are multiple intermediate states that can be candidates for modification by composing with the subfunctions, the nearest from the final state is selected. The rationale behind such a selection heuristic is that the later the modification done in a causal sequence, the lesser the disturbance caused in it (i.e., the lesser the distance the disturbance needs to be propagated before it reaches the output state).

Once a state in the causal behavior of the candidate design is selected for modification, then the template states in *f* and *g* are instantiated with the corresponding state-variable values in the selected states. The desired values in the selected state themselves are computed based on the overall functional differences and by regressing through the parameter relations annotating the transitions between the selected state and the final state. For instance, the magnitude of angular momentum in *state*₂ of the behavior of the candidate design of gyroscope follow-up can be determined to decrease if the output angular momentum rises due to fluctuations, given the specific qualitative relationships between state variables. Once the subfunctions are thus instantiated, the process involves designing for them and composing the new sub-behavior(s) with the behavior of the candidate design as per the relationships specified in the retrieved mechanism.

In the current design scenario, the subfunction IDEAL needs to design really has two parts (as it takes two inputs and produces one output): one that specifies the need for transferring angular momentum from the input location to the pivot location, and the other for transferring angular momentum from the output shaft location to the pivot location. In such cases where *f* or *g* have multiple GIVEN or multiple MAKES states, the behavior of the candidate design is

⁷These specific substances, their properties and values, and their locations are coming from the current design and are thus bound to the template specifications in the SBF representation of the feedback mechanism during matching.

first checked to see if any of the desired state transformations are already achieved by a segment. If so, the effective subfunctions to be designed for will have these states removed from the specification. That is, in the gyroscope follow-up design example, the first part of the subfunction formed is already designed for in the candidate design as the behavior segment $state_1 \rightarrow state_2$ (Figure 5.5(c)) achieves it. Therefore, in successfully instantiating the mechanism in the candidate design of gyroscope follow-up, IDEAL only needs to find a behavior (and a structure) that accomplishes the second part of the subfunction.⁸

Consider the concrete scenario from IDEAL in which it has the knowledge of a component (called *worm*) whose function is to transfer an input angular momentum to an output location with the magnitude proportional to the output component and the direction dependent on the direction of threading on the worm. This component reverses the direction of the input angular momentum. IDEAL retrieves that component because the desired part of the subfunction matches with its function. It substitutes the appropriate parameters in the behavior of the retrieved design (i.e., worm) to generate a behavior for the desired subfunction.

5.3.3 Composition of Behaviors

In order to complete the modification of the candidate design and generate new causal behaviors, IDEAL needs to compose the behavior of the candidate design with the behaviors for achieving the subfunctions. Therefore, in the next step, IDEAL composes that behavior (i.e., B_{22}) with the behavior of the candidate design (i.e., B_1) as per the specification of the causal relationships in the feedback mechanism (as in Figure 5.4(b)) to propose a behavior (shown in Figure 5.5(f)) for achieving the desired function. In general, it needs to compose two sub-behaviors, each corresponding to one of the subfunctions f and g . Finally, the changes in state variables due to the composition of behaviors need to be propagated forward in the causal behavior (until the end of the behavior or until the same state repeats) and any other behaviors in the model of the device that are dependent on the currently modified behavior. Note that the resulting modification in the design of gyroscope follow-up is a non-local modification because the topology of the candidate design changed. Thus the instantiation of GTMs can enable non-local modifications in device design.

⁸Of course, the existing partial design in the candidate design imposes constraints on what kind of design IDEAL can come up with for the part of the subfunction. For instance, the new component needs to be compatible with the worm-wheel if it were to be connected there.

Input: • M_1 , the SBF Model of the Design Analogue, and its Function F_1 .
 • F_2 , the desired function.
 • G , a GTM (retrieved by matching $F_2 \sim F_1$).

Output: • M_2 , the SBF Model of the new device that achieves F_2 .

Procedure:
begin
 (1) Select the behavior B_1 in M_1 relevant to F_1 .
 (2) Bind the initial & final states of B_1 to the appropriate GIVEN and MAKES states of the subfunctions f and g in G .
 (3) **if** \exists an unbound state variable in f or g
 then backtrace B_1 to find states in B_1 that may be modified,
 considering the bindings from step 2.
 (3.1) **if** \exists multiple candidate states for modification
 then Select the state that is nearest to the final state in B_1 .
 (3.2) Compute values of unbound state variables in f and g based on
 the selected state, $(F_2 \sim F_1)$, and PARAMETER-RELATIONS in B_1 .
 (4) **if** \exists multiple GIVEN or MAKES states in f or g
 then Check if $\exists b \in B_1$ that achieves the transformation
 from any of the GIVEN states to any of the MAKES states in f or g .
 (4.1) **if yes**
 then $f' =$ rest of the transformation in f .
 (i.e., $< (\text{GIVEN-states}(f) - \text{initial-state}(b)),$
 $(\text{MAKES-states}(f) - \text{final-state}(b)) >.$)
 $g' =$ rest of the transformation in g .
 (i.e., $< (\text{GIVEN-states}(g) - \text{initial-state}(b)),$
 $(\text{MAKES-states}(g) - \text{final-state}(b)) >.$)
 (5) Retrieve subdesigns for f' and g' .
 (5.1) **if** \exists no subdesigns for f' or g' **then** FAIL.
 (5.2) **else**
 (5.2.1) Adapt the retrieved subdesigns for f' and g' (if necessary).
 (5.2.2) Compose $B_{f'}$, the behavior for f' , and $B_{g'}$, the behavior for g' ,
 with B_1 as per the relationships in G .
 (5.2.3) Propagate the resulting changes in state variables forward in B_1
 and in the dependent behaviors in M_1 .
end.

Figure 5.6: IDEAL's Method for Instantiating A GTM

CHAPTER VI

PROBLEM REFORMULATION: USE OF GPPS

In the previous chapter, we described how the computational process of model-based analogy addresses different issues in the subtask of analogical transfer. The output of that subtask is a candidate solution to the target problem. More specifically, the proposal is in terms of a model for the candidate solution. The next issue is whether the candidate solution for the target problem really meets the constraints of the problem. The issues then become how to evaluate a candidate solution, and what knowledge may be needed to enable the evaluation and control the reasoning involved.

In general, there are two ways of evaluating a candidate design for the target problem: (1) verification by simulating a model of the device, which we call internal evaluation and (2) implementing the design in a real world and observing its behavior, which we call external evaluation. Since the output of the transfer step in the MBA process is a model for the target design, the process uses a model-based method for evaluating candidate designs internally. The crucial idea in this method is that the SBF model of the source analogue can be modified and simulated to verify if the proposed modifications indeed result in a solution for the target problem without actually making the modifications to the solution (i.e., the structure of the design). Designs may fail for several reasons. When the evaluation is done by simulating the model, if the MBA process finds that the design fails, the process tries to make a different modification to the source analogue and the control goes back to the transfer step.

When the evaluation is done externally by an oracle, if the design fails, there is an opportunity for the computational process to interact with the oracle and acquire feedback on the design failures. When design failures are observed in external evaluation, the interpretation (or understanding) of those design failures is a hard issue and subsequent redesign is yet another hard issue. Understanding the design failures is hard because it involves finding a cause for the failure in the context of the failing design. The model of the design alone is not sufficient for doing this task because it may not be complete. Therefore an issue is what knowledge might be needed to interpret design failures and finding their causes.

In this chapter, we will first describe how designs may be evaluated internally by simulating their SBF device models. Then we will describe how design failures in an external evaluation of designs can be understood, how their design problems can be reformulated and how the failed designs can be subsequently redesigned.

6.1 Internal Evaluation and Design Failures

We described in the previous chapter how the causal behaviors of a source design are modified according to the selected adaptation strategy to generate an SBF model for the desired design. Once the SBF model for the desired design is generated, the model is modified appropriately by propagating the changes to all parts of the behavior. Then the model is simulated to verify if the candidate design indeed achieves the desired function. Only then, the structure of the candidate design is modified to reflect the changes in the behavior and to generate a new design solution. In this section, we will focus on how localized changes are propagated throughout the behavior of a candidate design and how the design is internally evaluated.

In IDEAL, simulation of the effects of candidate modifications on the functions of the design involves tracing through the states and transitions in the model from the modified state to the final state by substituting new values for the parameters and checking if the desired function is achieved. Thus IDEAL uses the method of model revision and learns a new device model as a by-product. The types of knowledge required for revising the SBF model of the candidate design depend on the type of candidate modification to be executed on it. For instance, the method for model revision corresponding to the candidate modification of component replacement requires knowledge of how to *compose causal behaviors*, specifically, how to compose the causal behavior of the retrieved component with causal behavior segments from the candidate design. Similarly, each of the GTMs, the cascading, feedback and feedforward GTMs, specify different ways of revising the behaviors of candidate designs. Figure 6.1 shows the algorithm that IDEAL uses for revising the model of an old design and evaluating the proposed modifications by simulating the behavioral effects (i.e., the substeps of behavior modification and behavior verification). The original model-revision methods were developed outside the context of GTMs and were reported in (Goel, 1991b). In this research, we adapted and expanded them to deal with non-linear causal behaviors and new adaptation strategies such as the instantiation of GTMs. The function *update* in the algorithm updates the value of a given property in a given state or a given qualitative equation. At the end of SIMULATE, comparing the initial and final states of the new modified behavior with those in the desired function verifies whether the modifications worked.

Let us now trace through the algorithm given in Figure 6.1 in order to understand IDEAL's process of model revision. First, given a new state inserted or modified in the behavior of the candidate design, it stores all those properties whose values have changed from old to new. Then, it traces the behavior forward starting from the newly added state and propagates the changes to subsequent states and dependent behaviors.

The loop in the SIMULATE procedure essentially considers if the knowledge encapsulated in the next transition of the current state indicates whether any of the current changed properties are related to some substance or component property; if so, all those dependent properties are also modified and added to the set of changed properties (i.e., triples consisting of the property, its old value, and its new value). There are four different types of knowledge in the transition

Input: • Behavior in new design, B_{new} , in which a chosen property from the diagnosis has been changed.
 • Model of the old design, M_{old} .

Output: • Model of the new design, M_{new} , modified to be consistent with the changes in B_{new} .

Procedure:

initialize

$S_{diffs} = \{ (P_i, P_{i.value}^{old}, P_{i.value}^{new}), \text{ where } P_i \text{ is a property whose value } P_{i.value}^{old} \text{ in state } state_i^{old} \text{ has changed to } P_{i.value}^{new} \text{ in } state_i^{new} \text{ and } state_i^{old} \text{ and } state_i^{new} \text{ are respectively from } B_{old} \text{ and } B_{new}. \}$

visited-states = {};

SIMULATE ($state_i^{new}$, S_{diffs} , visited-states)

SIMULATE (state, S_{diffs} , visited-states)

current-state = state

LOOP

next-state = next-state (current-state);

IF next-state = NIL \cup current-state \in visited-states, THEN Exit LOOP

CASE :

(1) IF P_i is mentioned in next-state
 THEN update(P_i , next-state, current-state)

(2) IF there is a qualitative equation qe in state-next-transition(current-state),
 such that $P_l = f(P_i)$ where $(P_i, P_{i.value}^{old}, P_{i.value}^{new}) \in S_{diffs}$
 and P_l is mentioned in next-state
 THEN update(P_l , qe)
 $S_{diffs} = S_{diffs} \cup \{(P_l, P_{l.value}^{old}, P_{l.value}^{new})\}$

(3) IF there is a functional abstraction of a component E
 (i.e., Comp in USING-FUNCTION)
 and a condition on the substance properties on which E operates
 (i.e., $Condition_{SUB}$) in state-previous-transition(current-state)
 such that $P_i = P_l$, where
 P_l is a property of the substance in $Condition_{SUB}$, and
 $(P_i, P_{i.value}^{old}, P_{i.value}^{new}) \in S_{diffs}$
 THEN update(P_l , $Condition_{SUB}$)
 $S_{diffs} = S_{diffs} \cup \{(P_l, P_{l.value}^{old}, P_{l.value}^{new})\}$

(4) IF there is a pointer to a new behavior sequence B' such that
 the transition state-next-transition(current-state)
 affects a transition $trans$ of B'
 THEN spawn
 SIMULATE (transition-prev-state($trans$), S_{diffs} , visited-states)

END-CASE

visited-states = visited-states \cup {current-state}

current-state = next-state

goto LOOP

END-LOOP

Figure 6.1: IDEAL's Method for Model Revision

that may lead to revising the next state: (1) if the property is specified in the next state, then the next state is updated according to the changed value of the property; (2) if there is a qualitative equation in the next transition that relates a property in the current set of changed properties to a substance or component property, then the property value is updated and added to the set; (3) if a component contributes a function to the transition, and there is a condition on the property of a substance on which the component operates, and if that property is related in an equality relationship to a property in the current set of changed properties, then that property is also modified and added to the set; and (4) if the transition is dependent on a transition in another internal behavior of the device, then that behavior is also traversed and modified in the same way as the current one, starting from the previous state of the transition in that behavior.

At the end of the process described above, IDEAL would have modified all the behaviors by propagating the changes in the properties due to the application of a strategy. It then compares the output behaviors (i.e., functions) of the new design to the desired function to see if the design solution achieves the desired function. If IDEAL finds that the candidate design (i.e., the proposed design) delivers the desired functions, it considers that design as the target design and goes to the next step of learning from the source and the target designs. Suppose it finds that the design does not satisfy the constraints of the given problem. Then, it first tries an alternative adaptation strategy if one is available (i.e., for instance, applying the mechanism of cascading after trying component replacement to the same substructure in the source design). If no alternative strategies are applicable, then it tries to make a different modification to the source design if there are alternative modifications available that could result in reducing the functional differences (i.e., for instance, modifying the bulb instead of the battery in a simple electric circuit to deliver a different intensity of light). If no alternative modifications are possible, then it tries to adapt a different source design (i.e., for instance, given the problem of designing a high-acidity sulfuric acid cooler, it draws the analogy from the design of a high-acidity nitric acid cooler after having tried from the design of a low-acidity sulfuric acid cooler unsuccessfully). When there are no alternative source designs available or they do not lead to a target solution, IDEAL fails.

6.2 External Evaluation and Design Failures

Since IDEAL uses the SBF model of the candidate design to verify if it works and the SBF model may be incomplete, the system may not be able to detect some design failures by simulating the model. Also, some design failures may not be detected internally because the initial problem specifications may not always be complete, they may not indicate clearly the constraints from the environments in which the designs are intended to work, and the intended environments may themselves have changed from the time of problem specification to the time of design use. However, if an oracle presents the external feedback on failures of the design, IDEAL can understand the feedback and redesign the device. Figure 6.2 illustrates IDEAL's partial process

of model-based analogical design that centers around the evaluation task. A candidate design may fail due to a variety of reasons, e.g., it may not deliver the function desired of it, it may produce an undesirable behavior, etc. When a candidate design fails for some reason, several new tasks are set up: (1) understanding the feedback from evaluation of a candidate design (for instance, by forming a causal explanation for the undesired behaviors), (2) reformulating the problem (for instance, by discovering and incorporating new design constraints), and (3) redesigning the candidate design. When the candidate design fails, there are several ways of generating a new design: repair the failed candidate design to meet the new problem constraints or generate an alternative candidate design.

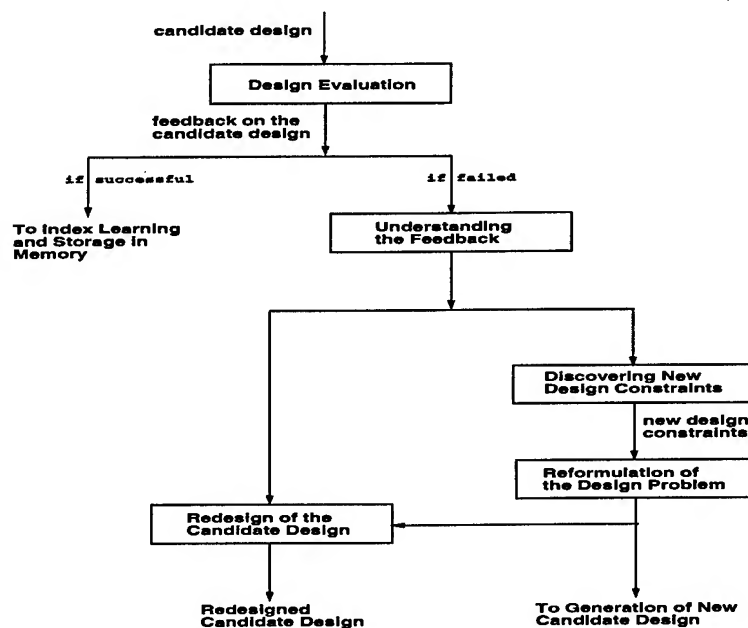


Figure 6.2: IDEAL's Partial Process (Evaluation and Redesign) of Analogical Design

We will consider the design of a coffee maker in order to illustrate this process model. This example has been adopted from (Prabhakar and Goel, 1992). Informally, the function of a coffee maker may be described as automatic brewing of hot coffee from coffee powder and hot water. Suppose that the designer generates a candidate design for achieving this function as illustrated in Figure 6.3(a). This design has two containers and a filter in between these two containers. Initially, Container-1 contains coffee powder. When hot water is added to it, the coffee powder dissolves in the water, forming a mixture of coffee powder, water and coffee decoction as indicated in Figure 6.3(a). Coffee decoction permeates through the filter and gets collected in container-2 as illustrated in Figure 6.3(b).

Suppose that the candidate design is implemented in a real device. Suppose further that an

evaluation of the device in a real environment shows that while this coffee maker achieves the function desired of it, two problems occur: (1) coffee-decoction in Container-2 is only lukewarm and (2) coffee-decoction does not stay warm in Container-2.

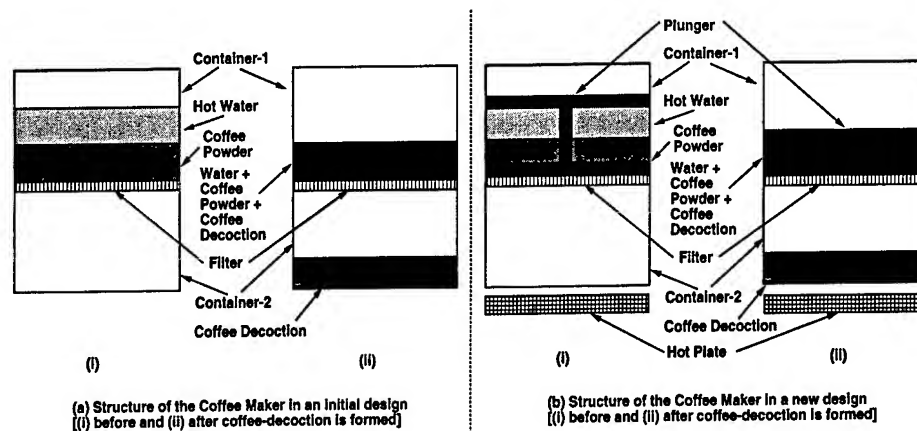


Figure 6.3: Failed and Redesigned Coffee Makers

Then the issue becomes how might a designer understand such feedback from evaluation and incorporate it in a new design for the coffee maker? One way a designer might understand this is by forming a causal explanation for the observed, undesired behaviors. The plausible causal explanations for the undesired behaviors in the coffee maker are (1) the coffee-decoction in Container-2 is only lukewarm because coffee making process is taking a long time during which heat escapes from the coffee-decoction in Container-1 to the environment and (2) coffee-decoction does not stay warm in Container-2 because it loses heat to the environment over time, which lowers its temperature.

Such causal explanations might enable a designer to discover new constraints on the design that introduce new variables and thereby change the design problem space. For instance, the causal explanations for the two undesired behaviors respectively suggest these two new constraints: (1) need to reduce the time for brewing coffee and (2) need to either reduce the time coffee stays in Container-2 (which is not under designer's control) or supply heat to Container-2 to compensate for the heat loss.

Given the reformulated problem with new constraints, a designer can either redesign the failed coffee maker or generate a new candidate design altogether. The failed design of the coffee maker may be redesigned, for example, by introducing a Plunger into Container-1 and a Hot-plate beneath Container-2, where these design modifications address the two newly discovered constraints. Figure 6.3(b) illustrates the redesigned coffee maker.

The above computational process requires two kinds of knowledge: (1) comprehension of how the coffee maker works and (2) comprehension of the process of flow of heat from hot object

to cold ones (i.e., the zeroth law of thermodynamics). In IDEAL, these two types of knowledge are represented in SBF language. Figures 6.3(a), 6.4(a) & 6.4(b) respectively illustrate the structure of the coffee maker, its function, and its internal causal behavior.

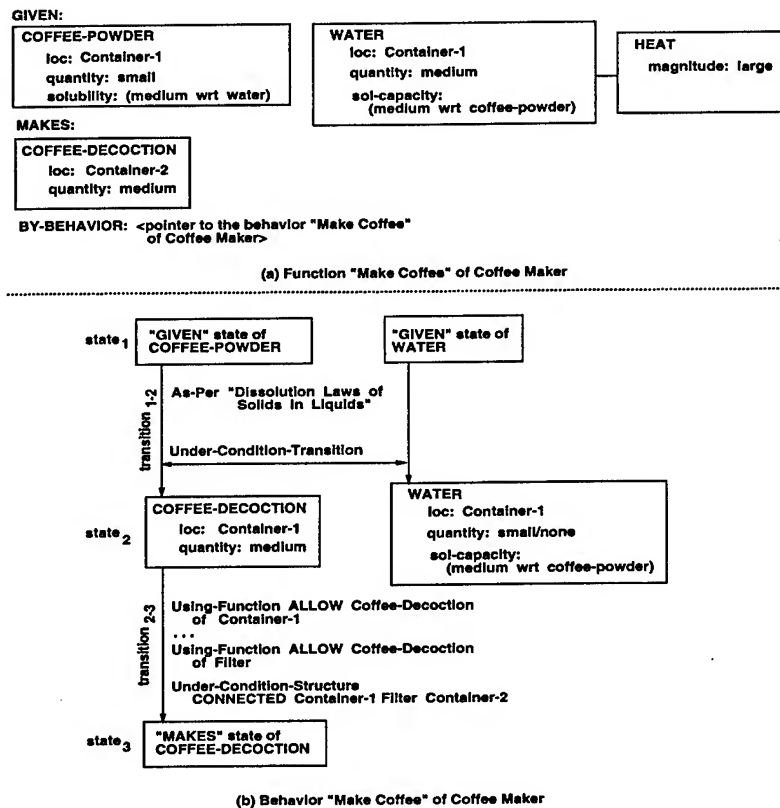


Figure 6.4: Function and Behavior of An Initial Design of Coffee Maker

6.2.1 Knowledge Acquisition: External Feedback on Design Failures

Since all failures cannot be determined automatically by means of simulating a model, knowledge of some of them may have to be acquired externally from an oracle. That is, if an oracle presents the external feedback on failures of the design, IDEAL can understand the feedback and redesign the device. One of the important issues in providing external feedback on designs concerns with the language in which the design failures may be described. In IDEAL, they are described in the same SBF language used to describe the comprehension of devices. Therefore, the primitives for describing the device failures include undesired states and undesired behavioral state transitions. For instance, the second failure of coffee maker mentioned earlier can be described as an undesired behavioral state transition illustrated in Figure 6.5, while the first

failure is described as an undesired state. Note that a true representation of the English description of the second failure needs to include a notion of time and a process extended over time. But IDEAL's representations are limited in that they do not represent time explicitly, and neither do they capture the notion of a process extending over time.

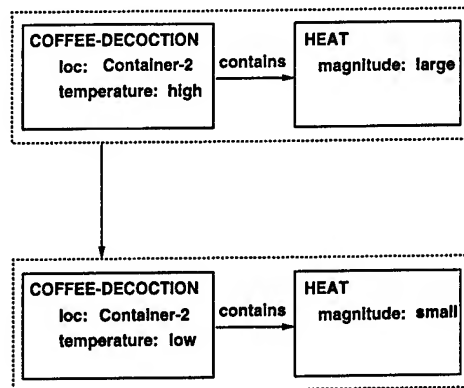


Figure 6.5: A Failure of Coffee Maker

6.3 Understanding Design Failures by Forming Causal Explanations

Once the design failures are provided externally, in order to redesign the device, IDEAL needs to first understand those failures in the context of the current design so that it can identify the causes for them and rectify the failures. Then the issue is how can IDEAL understand the feedback from the evaluation. It understands the feedback from the evaluation of a candidate design by forming causal explanations of the undesired behaviors described in the feedback. The question then is what knowledge of IDEAL enables it to make the inferences needed to form causal explanations for design failures. It uses the knowledge of generic physical processes for this task. Recall that IDEAL represents GPPs using the same SBF language that it uses for device models. It forms causal explanations of device failures by retrieving and instantiating the relevant GPPs.

6.3.1 Retrieval of GPPs

IDEAL accesses the GPPs by their behavioral abstractions. Since GPPs capture device-independent (or abstract) behaviors, IDEAL indexes them by their behavioral abstractions. Given the undesired behaviors in the candidate design, IDEAL uses them as a probe into its memory of GPPs and retrieves the physical processes whose behavioral abstractions match with the given behaviors. As mentioned above, it provides the SBF language for a user to specify

the undesired behaviors of a device in terms of states and state transitions. The use of this vocabulary allows it to access the relevant knowledge of physical processes. For instance, given the second failure of coffee maker design, IDEAL uses a description of the undesired behavior (Figure 6.5) as a probe into memory and retrieves the SBF representation of the GPP of heat flow (i.e., the zeroth law of thermodynamics) because the behavioral abstraction of the process (i.e., temperature change in a substance) matches with the specified undesired behavior (i.e., change of temperature of coffee-decoction from high to low). The retrieved model of the heat-flow process and its index are illustrated in Figure 6.6.

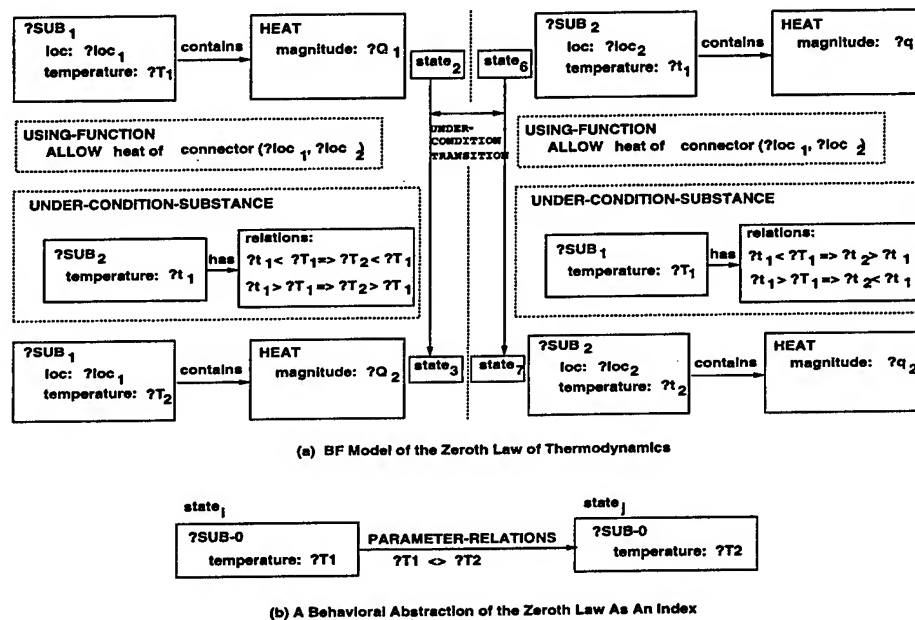


Figure 6.6: The GPP of Heat Flow

6.3.2 Instantiation of GPPs

Once a GPP is retrieved as match for the given undesired device behavior, IDEAL forms a causal explanation of why and how the given undesired behavior arises by instantiating the retrieved GPP in the context of the undesired behavior and the candidate design. The annotations on the transitions in the SBF representation of a GPP provide an explanation for the behavioral abstraction of the process used as an index for it. For instance, IDEAL instantiates the retrieved GPP of heat flow in the context of the second failure of coffee maker to form a specific causal explanation for the failure as illustrated in Figure 6.7. That is, it substitutes the specific substances and the values of their properties from the given undesired behavior into the retrieved process, according to the correspondences set up in retrieval.

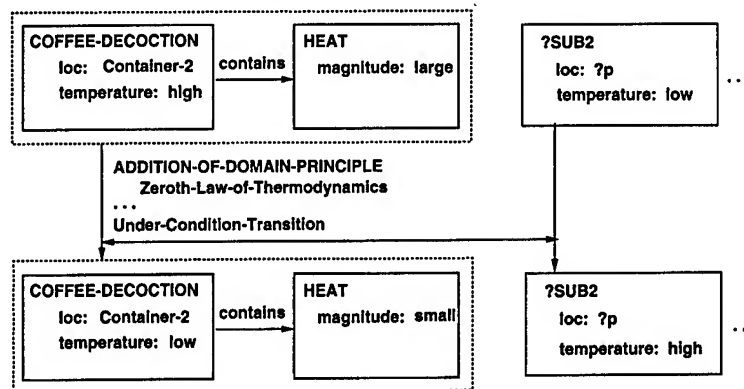


Figure 6.7: A Causal Explanation for One Failure of Coffee Maker

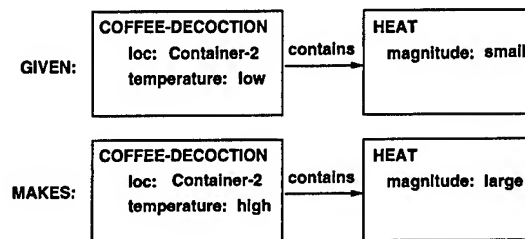


Figure 6.8: A New Function (subfunction) Desired of Coffee Maker

6.4 Problem Reformulation and Redesign

6.4.1 Discovery of New Constraints

Once the causal explanation for the failure has been formed, IDEAL can find the causes for the failure from the annotations on transitions in the explanation. It then hypothesizes new constraints on the design problem in the form of violation (or negation) of the causes for the failure. For instance, it finds from the causal explanation illustrated in Figure 6.7 that the cause for the second failure of coffee maker is due to the flow of heat from coffee-decoction in Container-2 to some substance outside the coffee maker (that is, in the device environment). Hence, it formulates a new constraint: avoid the loss of heat from coffee-decoction in Container-2.

In general, problem reformulation may involve one or more of the addition of new constraints and the deletion or modification of some old constraints. IDEAL reformulates the design problem of the coffee maker by adding the discovered constraints to the initial set of design constraints.

6.4.2 Designing for the New Problem

Once the design problem has been reformulated, IDEAL redesigns the failed design to generate a new candidate design. This involves two steps: (1) formation of behaviors for satisfying the new constraints (which might involve the addition of substructures in the design) and (2) composition

of the new behaviors with the behaviors of the failed design to generate a revised model.

For each new constraint, IDEAL postulates a new device function and posits it as a new design subproblem to be solved. For instance, it generates the new subfunction illustrated in Figure 6.8 for the new constraint it discovered from the explanation of the second failure of coffee maker. It generates this subfunction by reasoning that the transformation of the temperature of coffee-decoction from *high* to *low* (see Figure 6.7) needs to be reversed in order to eliminate the failure. IDEAL does so because the type of redesign it performs in this case is *compensatory redesign*. In this type of redesign, the goal is to introduce new behaviors into the device behavior such that they achieve the desired states from the undesired states. An alternative type of redesign is *corrective redesign* in which the current device behaviors are modified such that the undesired states or behaviors do not even occur. These types of redesign are discussed well in (Prabhakar and Goel, 1992).

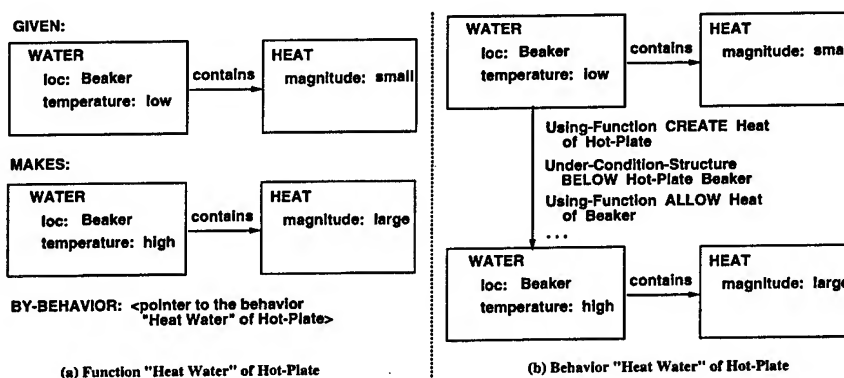


Figure 6.9: Function and Behavior of A Design Analogue of Hot-Plate

IDEAL then uses the subfunction as a probe into its memory of analogues to retrieve a matching design. Let us suppose that its memory contains the design of a HOT-PLATE whose function and behavior are illustrated in Figure 6.9. Informally, the function of this HOT-PLATE is to heat water contained in a Beaker. IDEAL indexes the design of HOT-PLATE by its function. Given the subfunction as a probe into the memory of analogues, it retrieves the designs whose functions match with the subfunction. In this example, it retrieves the design of HOT-PLATE because its function (Figure 6.9(a)) matches with the new subfunction to be achieved by the coffee maker (Figure 6.8). Since the retrieved design differs from the given subfunction in the substances involved, IDEAL adapts the retrieved design by a repair strategy called *substance substitution* (Goel, 1991a; and Section 5.2 in this chapter), that is, it substitutes the substances in the given subfunction for the corresponding substances in the function and behavior of the retrieved design. Similarly, it replaces the substance locations in the given subfunction for the corresponding ones in the retrieved design. By substituting coffee-decoction for water and

Container-2 for Beaker in the behavior of the HOT-PLATE, it generates a new behavior, illustrated in Figure 6.10, that satisfies the constraint specified in the subfunction (Figure 6.8). By this process, it proposes that the loss of heat from coffee-decoction in Container-2 can be compensated by supplying heat to coffee-decoction by introducing a HOT-PLATE below Container-2.

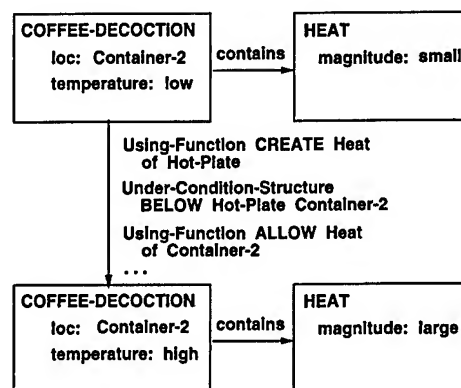


Figure 6.10: A new sub-behavior of Coffee Maker that achieves the new subfunction

6.4.3 Composition of Behaviors

Next, the behavior thus formed for the new subfunctions (Figure 6.10) is composed with the behavior of the failed design of coffee-maker (Figure 6.4(b)) to generate a behavior for the new coffee-maker design. In general, this process of model revision involves identifying the behavior segment (i.e., a transition and its preceding and succeeding states) or a state in the old behavior where the new behavior is to be added. In general, the composition may involve addition of old and new behaviors end-to-end (i.e., serial composition), in parallel, or merging of behavioral state transitions from both behaviors. IDEAL matches the initial and final states in the new behavior against the states in the old behavior, and selects a behavior segment where the new behavior can be added. For instance, it finds that the final state in the old behavior (Figure 6.4(b)), which is an undesired state, matches with the initial state of the new behavior, and hence it needs to add the new behavior at the end of the old behavior. For instance, the new behavior (Figure 6.10) is appended at the end of the behavior segment $state_2 \rightarrow state_3$ of the behavior of the failed coffee maker (Figure 6.4(b)). This process results in the revision of the behavior of failed coffee maker into the behavior of a new design of coffee maker, which is illustrated in Figure 6.11.

In a similar process, IDEAL redesigns the coffee maker to eliminate the first failure described earlier. It first forms a causal explanation for the failure by instantiating an SBF model of the heat exchange process and finds the cause for the failure as: the **flow-rate** of coffee-decoction

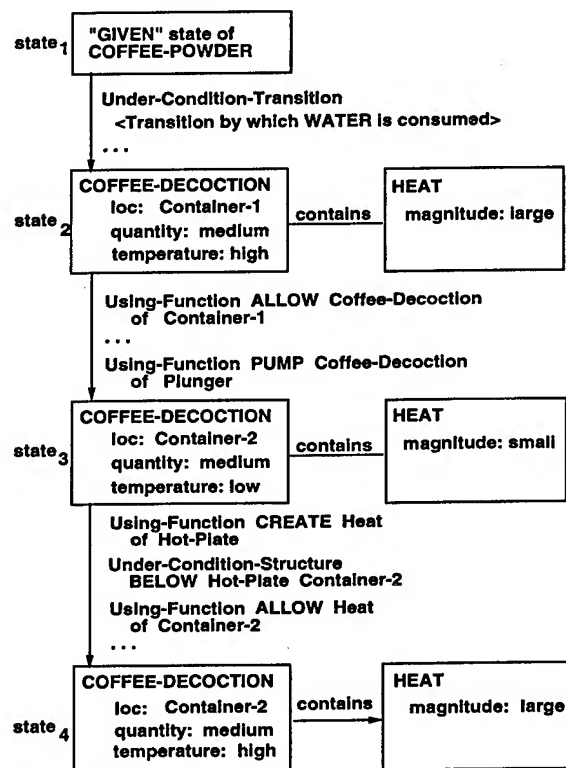


Figure 6.11: A Fragment of the Behavior of The Redesigned Coffee Maker

from Container-1 to Container-2 is *small*. It then postulates a new device subfunction which is to increase the flow-rate of coffee-decoction. It uses this subfunction to retrieve the design of a PLUNGER from its memory and adapts it to achieve the new subfunction of the coffee maker. Then it composes the adapted behavior of PLUNGER with the new behavior of coffee maker as illustrated in the behavior segment $state_2 \rightarrow state_3$ in Figure 6.11. In this case, the composition involves merging the new behavioral state transitions with the relevant state transitions in the old behavior, and modifying the rest of the old behavior accordingly. This redesign for the first failure of coffee maker is an example of corrective design.

IDEAL now evaluates the redesigned candidate design by qualitatively simulating (Goel and Prabhakar, 1991) the revised device model (Figure 6.11), generated by the above procedure. If the simulation reveals inconsistencies between the new functions of the device and its output behaviors, the redesign step needs to be repeated. In the case of the new coffee-maker, however, the simulation of the new SBF model (Figure 6.11) verifies that the candidate design indeed satisfies the new set of constraints (i.e., make coffee as well as keep it warm).

Nevertheless, the new design of coffee-maker may again fail when deployed in a real environment just like the initial design did, due to new interactions with the environment. And, in some cases, the environment may itself need to be adapted to suit the constraints of the device. A related project outside the realm of this thesis explores various issues in explicit modeling of environments and reasoning about their interactions with devices (Prabhakar, Goel and Bhatta, 1995).

CHAPTER VII

LEARNING DESIGN PATTERNS

In the previous two chapters we have discussed the use of design patterns in the tasks of analogical transfer and design modification, and the evaluation of candidate solution and its redesign. The next step in the MBA process is learning of design patterns from the source and target analogues. This step is triggered when a solution is available for the target problem, whether it is generated by the previous steps or is given directly by an oracle upon problem-solving failure. This step raises a number of issues such as: What might be learned from the target analogue that may be 'useful' for future problem solving? What might be abstracted from the target and source analogues? How might this abstraction occur?

Answers to these issues depend on the specific overall task being solved and the specific content theory of the SBF models in the task context. In the context of the design of physical devices and the theory of SBF models of devices, the pattern abstraction task involves a number of different learning tasks, each corresponding to a different type of design pattern that can be abstracted from a given source and target design analogues. We have explored the pattern abstraction task for learning of models of GPPs and GTMs. Each of these learning tasks takes as input the target design analogue and produces as output the respective type of model at a higher level of abstraction.

Our central hypothesis in addressing these two learning tasks is that the SBF models of specific devices provide the content for the abstraction, and they, together with the problem-solving context, provide the constraints for addressing the different issues in the abstraction process. In general, different methods such as similarity-based learning and explanation-based learning may be applicable for solving each of these learning tasks. Or, an integration of two or more methods may be more appropriate for the tasks. In this work, we have developed an approach that integrates model-based learning and similarity-based learning in which the model-based learning method is used first to focus on the relevant part and then the similarity-based learning is used to abstract over the selected part.

7.1 Issues in Learning by Abstraction

Abstraction over design experiences raises three important issues:

1. **The issue of relevance:** This is the issue of deciding what to abstract from an experience. We show that the problem-solving context in which learning occurs together with the

specification of the function of a new device and the hierarchical organization of the SBF model of the device helps in determining what to abstract from the model. Further, the SBF models lead to a typology of patterns of behavioral regularity over which the abstraction process can result in learning each type of abstract model. However, the computational process of MBA does not have a priori knowledge of what the regularities are, but rather, given two design experiences (one without an instance of the physical process or generic mechanism that might be learned and another with the instance of the respective model), it can discover the appropriate regularity by comparing and analyzing the given designs. Also, note that the SBF models define the dimensions and thus constrain the comparison of any two given behaviors (in the form of directed graphs that can include cycles).

2. **The issue of level of abstraction:** This is the issue of determining how far to abstract a chosen aspect of the device. We show that the similarities in the SBF models of the current design analogue and related design in a memory of analogues can help to determine how far to abstract. The knowledge of design objects, such as components and substances, help in determining the similarity between two design analogues.
3. **The issue of method selection:** This is the issue of deciding what methods to use for abstraction. This is especially relevant when there are multiple, specialized methods for learning different classes of abstractions. We show that a typology of device functions can help not only to determine what strategy to use but to determine whether physical processes (or principles) or models of prototypical devices (i.e., lower-level descriptions of physical processes) are formed. Furthermore, the typology of the patterns of regularity suggested by SBF models can help to determine what strategy to use.

We will now describe our model-based approach for each of the two learning tasks and present our explorations with different *situations* in which the learning tasks might occur.

7.2 Learning of GTMs

The task of learning generic teleological mechanisms takes as input a design analogue and forms an SBF representation of the GTM that is instantiated in the structure and behavior of the specific SBF model associated with the given design analogue. The input knowledge structure for the learning task is the SBF model of the given design analogue (which is case-specific) and the output knowledge structure is the case-independent model of a GTM. The learned generic mechanism is such that it is an abstraction over certain patterns of behavioral regularity (explained later) observed in the given SBF model and the model of the most similar design in analogue memory (i.e., the source design analogue). A formal characterization of the learning task is shown in Figure 7.1. We have explored the learning of different generic mechanisms such as cascading, feedback, feedforward, and device composition mechanisms.

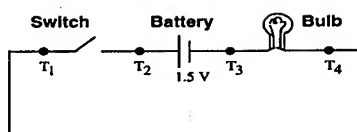
-
- Input:** • Design Analogue [consisting of design problem (i.e., function), solution (i.e., structure), and explanation (i.e., SBF model)].
e.g., design of a sulfuric acid cooler.
- Output:** • Generic teleological mechanism (represented in SBF language).
e.g., cascading mechanism.
- Method:** • Model-based learning and similarity-based learning.
e.g., function of a design determines what parts of the experience to focus on, and similarities between the given design analogue and the source design analogue determine how far to abstract the focused part.
- Knowledge:** • Typology of primitive functions in the domain.
e.g., ALLOW, PUMP.
• Typology of functions in the domain (consisting of primitive functions).
e.g., substance-parameter transformation.
• Typology of behavioral regularities in SBF representations.
e.g., causal loops, forks, and joins.
• Substances in the domain.
e.g., nitric acid, water.
• Components in the domain.
e.g., pipe, chamber.
• Past design analogues in memory.
e.g., design of a nitric acid cooler.

Figure 7.1: Task of Learning Generic Teleological Mechanisms from Design Analogues

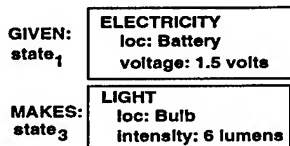
7.2.1 An Illustrative Learning Task: Learning of Cascading GTM

Since in our theory, learning is closely integrated with problem solving and memory, it is necessary to preface our description of the learning method with the appropriate problem-solving context and the state of memory. Suppose, for instance, IDEAL's analogue memory has the design of a 1.5-volt electric circuit (EC1.5) reproduced in Figure 7.2 from Chapter 1. Figure 7.2(a) shows the structure of the circuit schematically. Figure 7.2(b) shows the function "Produce Light" of EC1.5. Informally, the function specifies that the device takes as input electricity with a voltage of 1.5 volts in Battery and gives as output light with an intensity of 6 lumens in Bulb. Figure 7.2(c) shows the internal causal behavior that explains how electricity in Battery is transformed into light in Bulb. The causal behaviors can be specified at different levels of detail. For instance, $state_1$ is an aggregation of a sequence of several states and state transitions at a different level as shown in Figure 7.2(d).

Let us now consider the scenario where IDEAL is presented with a problem of designing a 3-volt electric circuit (EC3). The problem specifies that the desired function of the device is to produce light of intensity 12 lumens in the bulb when the switch is closed, given an electricity with a voltage of 3 volts in the battery. In addition, it also specifies a structural constraint



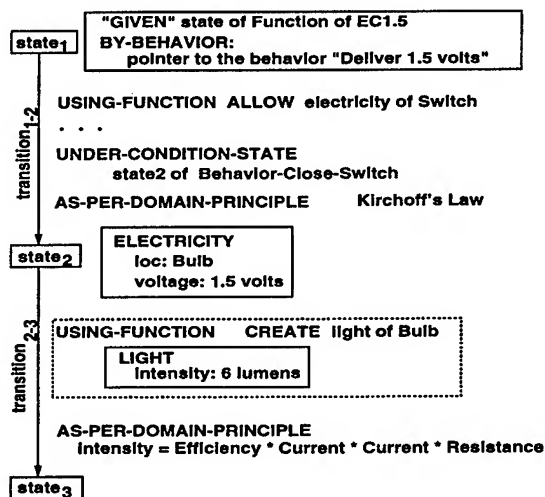
(a) 1.5-volt Electric Circuit (EC1.5)



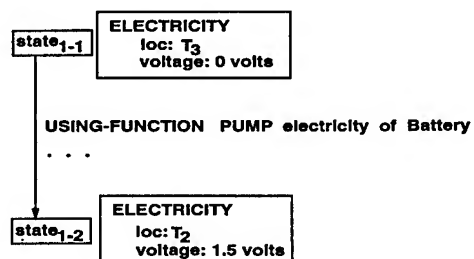
STIMULUS: Force on Switch

BY-BEHAVIOR: pointer to the behavior "Produce Light"

(b) Function "Produce Light" of EC1.5



(c) Behavior "Produce Light" of EC1.5

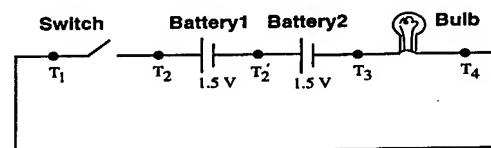


(d) Behavior "Deliver 1.5 volts" of Battery

Note: All locations are with reference to components in this design. All labels for states and transitions are also local to this design.

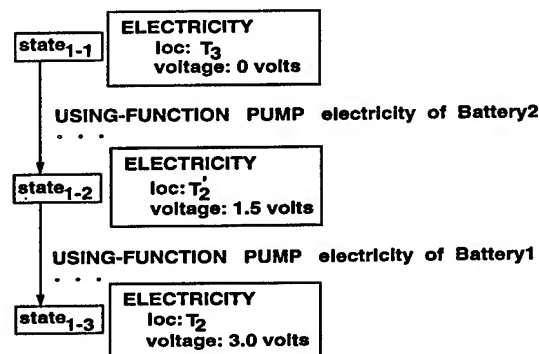
Figure 7.2: Design of A 1.5-volt Electric Circuit (EC1.5)

that the design cannot have a single 3-volt battery. IDEAL retrieves the design analogue EC1.5 because the given functional specification is similar to the function of EC1.5. However, IDEAL may know only how to replace a component in a past design to solve the current problem. The component-replacement strategy specifies how to replace the component that is responsible for the functional difference by a new component that reduces the functional difference and thus enables the overall device to deliver the desired function. In such cases, IDEAL fails to solve the current problem due to the structural constraint specified. Then an oracle can interact with IDEAL and provide feedback so that it can learn. Suppose that an oracle presents the correct solution that both delivers the desired function and satisfies the structural constraint (the schematic of the structure of the new device is shown in Figure 7.3(a)). Then IDEAL first learns how the new device behaves (a segment is shown in Figure 7.3(b)) by revising the behavior of EC1.5. This problem-solving context enables IDEAL to focus on the substructure that delivers the required voltage for comparing with the corresponding substructure in the source design EC1.5. By abstracting over the pattern of regularity in the behavioral segments of the corresponding substructures in the two designs, it learns the cascading mechanism. We will now focus on the learning of the cascading mechanism.



(a) 3-volt Electric Circuit (EC3)

Note: The behavior "Produce Light" of EC3 at toplevel is similar to that of EC1.5 except for the parameter values of voltage and intensity. Also, the slot BY-BEHAVIOR in state1 points to the behavior "Deliver 3 volts" of Battery shown in Figure (b).



(b) Behavior "Deliver 3 volts" of Battery

Figure 7.3: Design of A 3-volt Electric Circuit (EC3)

7.2.2 Different Interaction Conditions and Learning Situations

As mentioned above, when IDEAL fails, an oracle can provide it with different kinds of feedback so that it can learn. We have in fact explored several different learning situations here. The external feedback from an oracle may have different kinds of information: (1) all the three constituents of a design analogue, i.e., the function, structure and behavior for the new design; (2) only the function and structure for the new design; or (3) only the function and the localized substructure in the context of the failure (i.e., a solution for the specific, local adaptation goal). Alternatively, IDEAL may not even fail but rather produce a design for the given problem by exploring with the knowledge of alternative strategies such as the component-addition strategy. In this case, IDEAL may take more modification steps than necessary.

In this chapter, we assume that the oracle presents the desired design and its SBF model, the simplest of the interaction conditions. We will consider the other interaction conditions in the next chapter, and describe how the SBF model of the new design itself is first learned. Given the SBF models of the retrieved design and the desired design, IDEAL compares the respective behaviors in the two models and forms any generic mechanisms it can discover.

7.2.3 The Model-Based Learning Method

The learning method is model-based in that the SBF models of the design analogues provide the content for abstracting over the patterns of regularity in the device structure and device behavior. The method is shown in Figure 7.4. The representation vocabulary of the SBF models further defines the dimensions along which two behaviors can be compared and leads to several classes of regularity based on the "cause-effect" relationships between behavior segments.¹ For instance, behavioral states, transitions, and behavioral segments (state-transition-state) are some dimensions at a top level along which two behaviors can be compared. Within comparing two behaviors along these dimensions a few of the lower level dimensions are substances and components, their properties and values, and primitive functions and their ranges of transformation. A few patterns of regularity in device behaviors, say, B_1 and B_2 , are illustrated in Figure 7.5; B_1 is the behavior of an available or candidate design (i.e., a source design analogue) and B_2 is the behavior of the desired design (i.e., the target design analogue). In the following discussion, F_1 denotes the function of the candidate design and F_2 that of the desired design.

The learning method first traverses the two focused behaviors and compares them for similarity. When the behavior of the source design analogue (B_1) matches with (or is similar to) some segment in the new device behavior (B_2), then there is an opportunity for IDEAL to learn a generic mechanism that specifies how to modify a behavior like B_1 to get a behavior like B_2 that achieves the function like F_2 . Suppose that $B_2 = B_{21} + B_{22}$, a composition of two behavior segments, and that B_1 matches with B_{21} . Under such conditions, IDEAL can form a generic

¹A behavior segment is a partial sequence of states and transitions in a behavior. The smallest behavior segment will have just two states and a transition between them.

Input: • F_1 , Function of Device1 and M_1 , its SBF Model.
 • F_2 , Function of Device2 and M_2 , its SBF Model.

Output: • G , the SBF representation of a GTM (whose instance is in Device2's model).

Assumptions: 1. B_1 , the internal behavior of F_1 , is without an instance of a GTM.
 2. B_2 , the internal behavior of F_2 , has an instance of only one GTM.
 3. $\text{Length}(B_1) \leq \text{Length}(B_2)$, where Length is measured in terms of number of state transitions.

Procedure:
begin
 (1) Select internal behavior segments B_1 & B_2 respectively in M_1 & M_2 relevant to F_1 & F_2 and ($F_2 \sim F_1$).
 (2) Compare B_1 & B_2 to find ($B_2 \sim B_1$) along all dimensions of the representation by traversing B_1 & B_2 state by state.
Termination conditions for traversal:
 (B_2 reaches FINAL-STATE) \cup (a state is revisited).
 (2.1) if $B_1 \notin B_2$ then NO LEARNING.
 (2.2) else
 (2.2.1) Separate matching segments and unmatching segments in B_2 into 3 sets.
 $B_{21} = \{B_{2i} \mid B_{2i} \text{ matches } B_1\}$
 $B_{22-1} = \{B_{2j} \mid B_{2j} \text{ precedes } B_{21} \text{ in } B_2\}$
 $B_{22-2} = \{B_{2j} \mid B_{2j} \text{ succeeds } B_{21} \text{ in } B_2\}$
 (2.2.2) Form a functional abstraction over each of these behavior segments, and record the causal relationships among them.
 $f = \langle \text{INITIAL-STATES}(B_{22-1}), \text{FINAL-STATES}(B_{22-1}) \rangle$
 $g = \langle \text{INITIAL-STATES}(B_{22-2}), \text{FINAL-STATES}(B_{22-2}) \rangle$
 if $|B_{21}| > 1$
 then $F_2 = f + n * F_1 + g$, where $n = |F_2| / |F_1|$
 else $F_2 = f + F_1 + g$
 (where $a + b$ denotes composition of behaviors that achieve functions a, b .)
 (2.2.3) Generalize all the states in these subfunctions as per the similarities and differences in B_1 & B_2 .
end.

Figure 7.4: A model-based method for abstracting GTMs over regularities in design analogues

mechanism only on the basis of its analysis of differences between B_1 and B_2 , in particular, the relationships between B_1 and B_{22} . IDEAL's hypothesis is that the difference in the functions of the two devices ($F_2 - F_1$) can be attributed to the difference in the behaviors ($B_2 - B_1$, which is the additional behavior B_{22}) and the relationships between B_1 and B_{22} . The basic idea is to decompose B_2 in terms of B_1 as much as possible because that informs how to modify a design that achieves B_1 into achieving B_2 . While comparing B_1 and B_2 , when the traversal of B_1 reaches the end (i.e., its final state), then B_1 is compared again from its initial state with the remaining

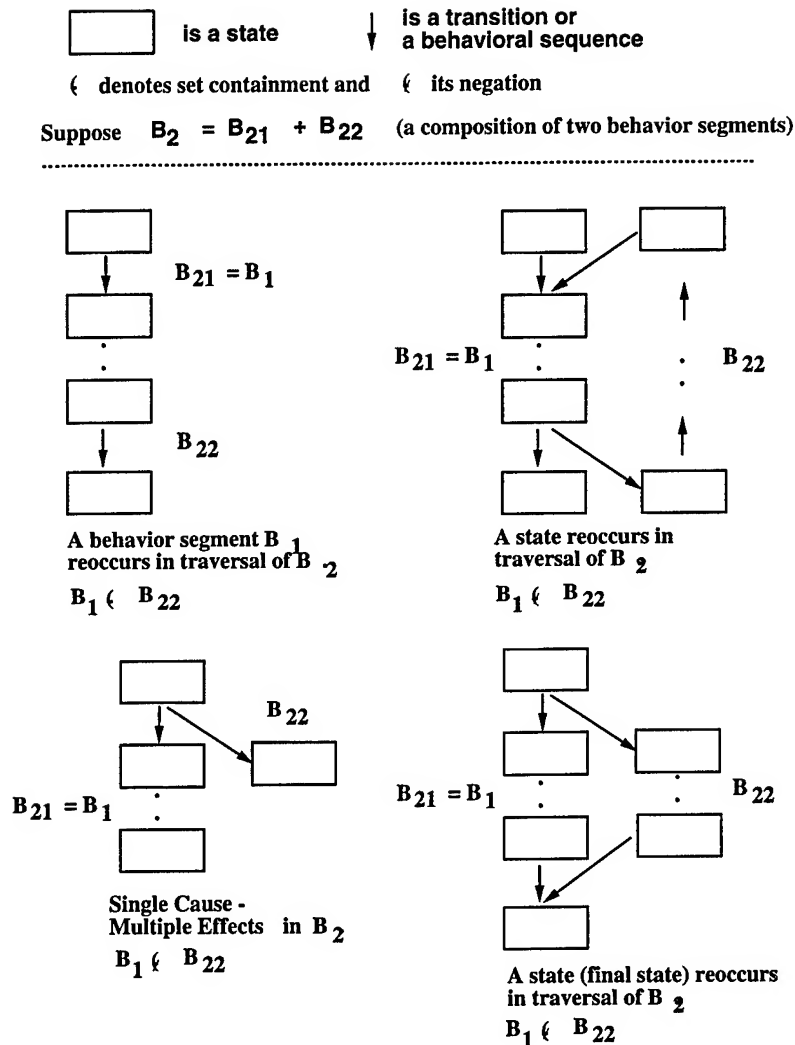


Figure 7.5: A Few Patterns of Regularity in Device Behaviors

states and state-transitions in B_2 (i.e., B_{22} after the first traversal of B_1 when B_1 matches with B_{21}). And, this process is repeated until all the states of B_2 have been traversed at least once. While traversing the two focused behaviors, IDEAL can recognize the end of a behavior, recurrence of states, single cause-multiple effects (i.e., a fork in the directed graph representation of causal behavior), and multiple causes-single effect (i.e., a join in the directed graph), and repetition of a behavior segment, without an explicit a priori knowledge of them. Recognition of these basic patterns is necessary for a terminable traversal of a behavior! From these basic patterns, IDEAL discovers the regularities in the relationships between B_1 and B_{22} and forms mechanisms such as cascading, feedback, and feedforward. It then abstracts over the specifics of these relationships so that the learned mechanism is useful in different problem-solving contexts.

In addition, in order for a new mechanism to be useful, IDEAL needs to identify the appli-

cability conditions for the mechanism. Because IDEAL believes that the relationships between B_1 and B_{22} to be responsible for the difference in the candidate and desired functions, it forms the decomposability condition on the desired function as one of the applicability conditions. For instance, when a desired function, F_2 , is specified, and a candidate design delivers F_1 , one applicability condition for using “a” generic mechanism is to check if F_2 can be decomposed in terms of F_1 and any other subfunctions and precisely what those additional subfunctions are. Hence it finds the subfunctions besides the candidate function (F_1) that the desired function can be decomposed into (i.e., $F_2 = f + n * F_1 + g$ or $F_2 = f + n * F_1$ or $F_2 = n * F_1 + g$ where $n (\geq 1)$ is the number of different behavior segments in B_2 that match B_1). Since, in general, segments in B_{22} can be distributed partly preceding B_{21} and partly succeeding B_{21} , there can only be at most two subfunctions other than F_1 (or multiples of it)—one subfunction f that is an abstraction over the behavior segments that precede B_{21} (i.e., B_{22-1}) and the other subfunction g that is an abstraction over the behavior segments that succeed B_{21} (i.e., B_{22-2}). By tracing B_{22} back from the initial state of B_{21} and tracing it forward from the final state of B_{21} in B_2 , IDEAL can identify the segments B_{22-1} and B_{22-2} and find their functional abstractions f and g in order to learn an applicability condition for the new mechanism. IDEAL analyzes in this manner because it would enable discrimination among competing mechanisms in terms of these subfunctions of a desired function, while using the mechanisms in later problem solving. For instance, f and g would be different for feedback and feedforward mechanisms. Thus IDEAL describes all the generic mechanisms it learns in a uniform representation, that is, in terms of relationships between candidate and desired functions and the corresponding behaviors. In sum, a generic mechanism specifies in a pattern-like, device-independent manner, yet using the SBF representation language how to compose the candidate behavior (that achieves F_1) with the behaviors that achieve the subfunctions f and g to generate a behavior that achieves the desired function F_2 where F_2 can be decomposed into f , F_1 (or multiples of it), and g .

In addition to the higher level dimensions of comparison described above, there are two important variables of interest that are provided by the representation vocabulary of the SBF models: primitive functions and their ranges of transformation in each behavior segment (r) and the number of different behavior segments in B_2 that match with B_1 (n). The possible outcomes of interest for comparing r 's are whether the range of transformation for B_1 is *equal* to the range of transformation in each matching behavior segment in B_2 or they are *not equal*, and the values of interest for n are whether $n = 1$ or $n > 1$.² Given the task of learning from two design analogues and the above values for the two variables, four different situations are possible as shown in Table 7.1. An SBF representation of the cascading mechanism can be learned when the regularity in the source and target design analogues is as in situation 2.

In the problem-solving scenario of designing EC3 from EC1.5, the problem-solving context indicates that the behavioral segments to focus on for learning are those that correspond to

²Only these two values of n are of interest because they define qualitatively different kinds of modifications to B_1 in order to make it into B_2 .

Table 7.1: Situations of Regularity Between Similar Behavior Segments in Two Design Analogues

Situation	Range of Input-Output Transformation in both behavior segments, r	Number of Repetitions of B_1 in B_2 , n	What can be Learned?
1.	equal	$n = 1$	Abstracted Behavior-Function relationships between B_1 and B_2 . (e.g., generic mechanisms such as feedback, feedforward, and device composition)
2.	equal	$n > 1$	Abstraction over n . (e.g., cascading mechanism)
3.	not equal	$n = 1$	Abstraction over r . (e.g., prototypical device models)
4.	not equal	$n > 1$	None due to lack of regularity.

the function of **Battery** in the two designs, EC1.5 and EC3. They are respectively $state_{1-1} \rightarrow state_{1-2}$ shown in Figure 7.2(d) (i.e., B_1) and $state_{1-1} \rightarrow state_{1-3}$ shown in Figure 7.3(b) (i.e., B_2). Applying the above learning method, it is easy to identify that the regularity is as in situation 2 shown in Table 7.1. That is, B_1 matches with more than one behavior segment, namely, $state_{1-1} \rightarrow state_{1-2}$ and $state_{1-2} \rightarrow state_{1-3}$, in B_2 ; and the range of parameter transformation in B_1 as well as each matching behavior segment from B_2 is same (i.e., 1.5 volts and the primitive function is "PUMP electricity"). Abstracting over the number of repetitions of B_1 in B_2 (which is > 1) and variablizing the range of parameter transformation, IDEAL hypothesizes a generic mechanism that would help in a problem-solving context similar to the current one. The model of the *hypothesized* cascading mechanism and its index are shown in Figure 7.6 (representations in (a) and the shaded region of (b)); the functional difference that the cascading mechanism reduces is the index for the mechanism.³ In addition, the index for the cascading mechanism consists also of the decomposability condition, which is $F_2 = n * F_1$; since there are no preceding or succeeding segments in B_2 that do not match with B_1 , both f and g are *null* functions.

7.2.4 Incremental Revision of the Learned Mechanisms

Forming an initial hypothesis for a mechanism based on the current source and target design analogues is itself a difficult task and involves the kind of comparative analysis described and illustrated above. But revising an already formed hypothesis based on the new set of source and target design analogues is a harder task and brings up several new issues: (1) at any given time, will there be only one description of the mechanism or several alternative descriptions held? (2) if only one description is held, how is a new hypothesis unified with the existing one? (3) if multiple descriptions are held, how is it decided whether the new hypothesis gets added as yet another alternative description or it gets unified with one of the existing descriptions? (4) how is a new hypothesis unified with an existing one? (5) how many problem-solving situations (i.e., training examples) does it take to arrive at a stable description of a generic mechanism (or multiple stable descriptions as the case may be)? and (6) whether the order of problem-solving situations presented matters, and if so, how does it affect learning?

In this work we take the approach of maintaining minimal number of alternative descriptions for a mechanism at any time. The decision of whether the new hypothesis is unified with an existing one or is held as yet another alternative is predicated upon whether their indices can be unified seamlessly. By seamless unification we mean that the two indices are transformed into a single entity (and not merely as a disjunctive set). Since the generic mechanisms are indexed by the functional differences between the candidate design and the desired design, and by the decomposability condition of the desired function, the representation vocabulary of the SBF language used provides a way of determining (and unifying if possible) whether two given indices

³A new piece of knowledge learned is futile unless its applicability conditions (or *indices* as we call them) are also learned.

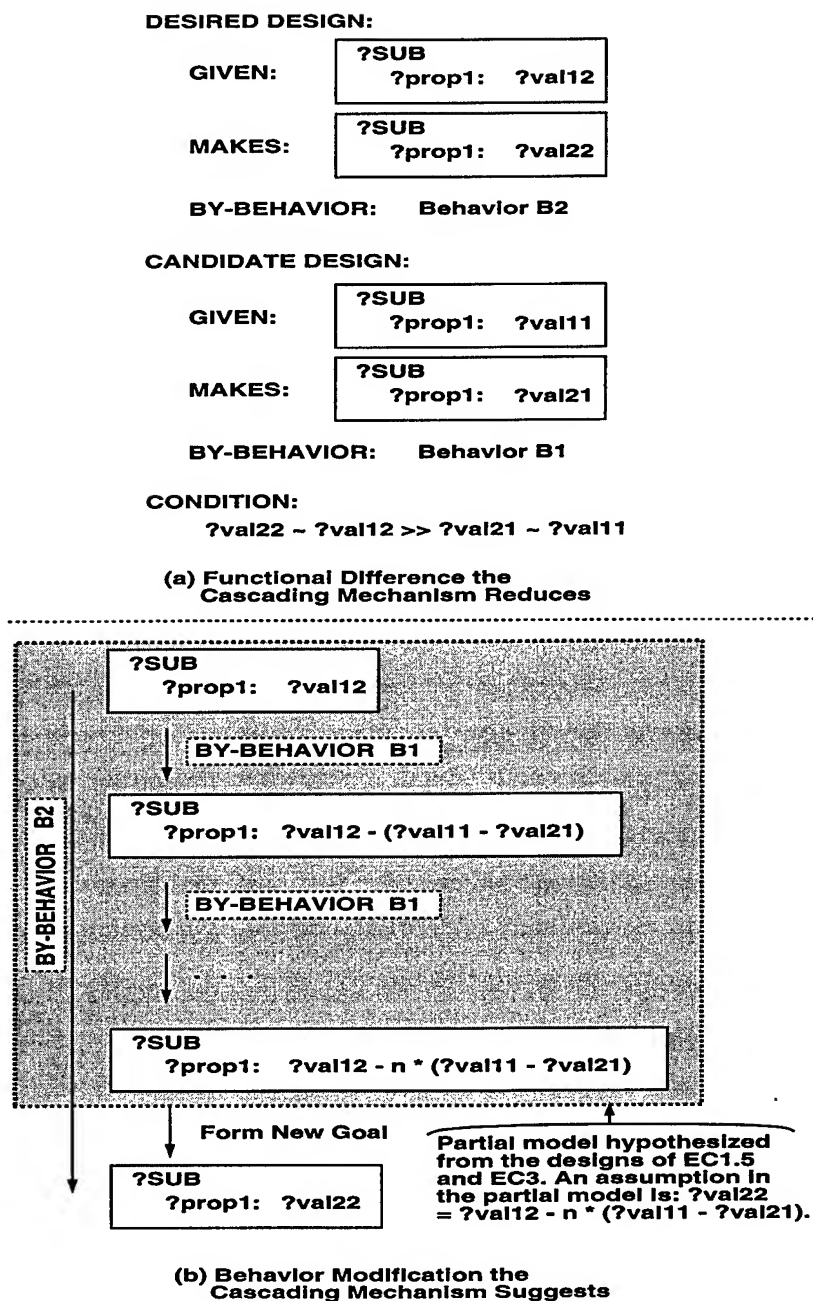


Figure 7.6: A Complete Description of the Cascading Mechanism in SBF Representation

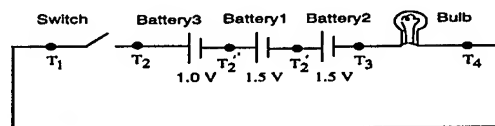
can be unified seamlessly. Suppose that two indices for two different descriptions of a mechanism specify the decomposability conditions as $F_2 = f_1 + n_1 * F_1 + g_1$ and $F_2 = f_2 + n_2 * F_1 + g_2$. Then, in order for these indices to be unifiable, the corresponding constituents (f_1 and f_2 , n_1 and n_2 , and g_1 and g_2) must all be unifiable. We should note that a *null* constituent unifies with a *non-null* constituent and their unified description will be same as the non-null constituent. Since the f 's and g 's are functional abstractions and are represented using the SBF language, it is easy to determine whether two given functions can be unified into a single function. The unifiability of two functions can be recursively defined by the conjunctive unifiability of the corresponding constituents of the functions. That is, two functions can be unified if their input states can be unified as well as their output states. Similarly, two states can be unified if both are of the same type (substance state or component state) and both refer to the same substance or the same component (or variablized place-holders for them). When unifying two substances or components, if both of them specify a property, then the corresponding values are unified (that is, they are replaced by a single value that is more general than each individual value and that covers both). As far as n 's are concerned, since our theory makes a commitment to distinguish between $n = 1$ and $n > 1$, considering them to be qualitatively different, the same commitment has to be respected in any inference that concerns n . Therefore, two n 's are unifiable if both are 1 or both are > 1 . Once the given two indices are unified, it is easy to unify the parts of the mechanism that specify the behavior modification along the same lines as the functions are unified.

An analysis of our learning method suggests that the order of design problem-solving situations only affects how fast a stable description of a generic mechanism can be arrived at. For any given order of presentation, the number of training situations needed to arrive at a stable description of a generic mechanism is different for different types of generic mechanisms. For instance, learning feedback mechanism (all different types of feedback) requires more training situations than learning cascading mechanism. Overall, however, even in the worst case, it requires only a few (< 10) design situations to learn the most complex type of generic mechanism; this is because the case-specific SBF models of the design analogues together with the problem-solving context constrain the learning process by providing focus.

Let us now consider the revision of the initially hypothesized model of the cascading mechanism in order to illustrate our theory. That learned model of the cascading mechanism is a partial model in that it specifies that it can be used only when the larger functionality desired is an *integral multiple* of the smaller, candidate functionality. IDEAL can revise the hypothesized model into a more complete one when it solves a new design problem whose model has a *behavioral pattern* that is an instance of the complete cascading mechanism. For instance, such a design problem will specify a desired function and a structural constraint as follows. The desired function is to produce light of intensity 16 lumens in the bulb as output when the switch is closed, given as input an electricity with a voltage of 4 volts in the battery. And, the structural constraint is that there is no battery which can deliver electricity with a voltage of 4

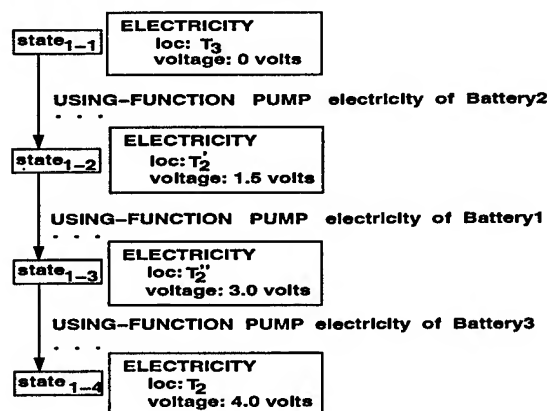
volts but there are batteries that can deliver electricity of 1.5 volts and one battery of 1 volt. The solution to this problem will have a structural pattern where two 1.5-volt batteries and one 1-volt battery are cascaded. Thus acquiring a complete model of a generic mechanism may involve solving a number of design problems incrementally.

Given this new problem, both the designs EC1.5 and EC3 are retrieved because both of them are functionally similar to the new desired function. Suppose that EC1.5 is selected for transfer and modification. Both the component-replacement and the learned cascading mechanism are applicable in this situation in order to modify the retrieved design; but the latter is more applicable than the former because of the structural constraints specified. Nevertheless, since the learned cascading mechanism is only partial, applying it in the current situation does not lead to a successful design for the new problem. Recall that the learned cascading mechanism is partial: it does not specify that a candidate design which achieves a smaller function F_1 can be replicated as many times as possible (say, n) so that the composite function is \leq the desired function F_2 and that an additional design can be composed to achieve the residual function $(F_2 - n * F_1)$. Under such failure conditions, if the correct solution (Figure 7.7) to the new problem is provided to IDEAL (either directly or when knowledge of component addition is available IDEAL itself generates a design by using an available component repetitively), it can revise the previously learned model of the cascading mechanism into a more complete one.



(a) 4-volt Electric Circuit (EC4)

Note: The behavior "Produce Light" of EC4 at toplevel is similar to that of EC1.5 except for the parameter values of voltage and intensity. Also, the slot BY-BEHAVIOR in state1 points to the behavior "Deliver 4 volts" of Battery shown in Figure (b).



(b) Behavior "Deliver 4 volts" of Battery

Figure 7.7: Design of A 4-volt Electric Circuit (EC4)

IDEAL's revision of a learned mechanism involves first, forming a new hypothesis from the current source and target design analogues using the method described in Section 7.2.3, and then unifying (merging or assimilating) the new hypothesis with the so far learned mechanism. Following the learning method described, IDEAL focusses on the behavioral segments that correspond to the function of **Battery** in the two designs EC1.5 and EC4, and finds by comparing the two that the regularity is as in situation 2 (Table 7.1). That is, B_1 (the behavior segment in EC1.5) matches with more than one segment in B_2 (the behavior segment in EC4). It also finds that there is a behavior segment in B_2 that does not match with B_1 and a functional abstraction of that segment (i.e., g) needs to be noted in the decomposability condition of the desired function F_2 . Hence, the newly hypothesized model of cascading mechanism specifies a more general decomposability condition, which is $F_2 = n * F_1 + g$; since there is no preceding segment in B_2 that does not match with B_1 , f is a *null* function. Note that IDEAL could have found f to be the functional abstraction over the preceding segment in B_2 and g to be *null* if the new design were as shown in Figure 7.8. (The two designs in Figures 7.7(a) & 7.8 differ in the way 1-volt battery is composed with the other two batteries.) Once a new model of the cascading mechanism is hypothesized, IDEAL compares it with the previous model and unifies the two (as per the process described in the beginning of this section); the result is a model as shown in Figure 7.6 (including both shaded and unshaded representations shown in the figure).

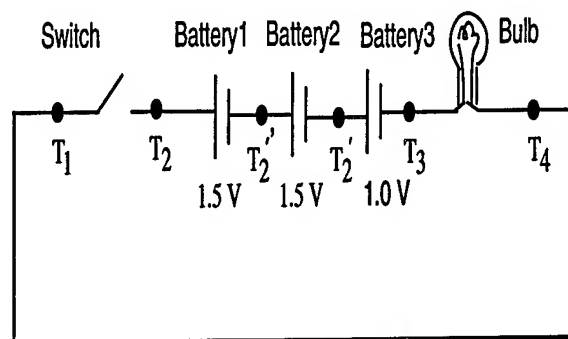


Figure 7.8: An Alternative Design of the 4-volt Electric Circuit (alternative with respect to the configuration of batteries)

7.2.5 Learning of Feedback GTM

Let us now consider a scenario in which IDEAL learns one type of feedback mechanism. The scenario is that given two designs from the domain of electronic circuits, one without an instance of the feedback and the other with an instance, IDEAL compares their models and abstracts over a pattern of regularity to learn the feedback mechanism. We have described the same scenario of learning feedback GTM in Chapter 2. But there is a subtle difference in the representation

of fluctuations of output values. In that scenario, we assumed that the fluctuations are already specified in qualitative, abstract terms (e.g., large *vs.* small) in the representation of the specific designs while in the current scenario we relax that assumption. A more realistic representation of specifying fluctuations in device design is in terms of a specification of quantitative/numerical tolerance limits (e.g., the output property value is 100 ± 10). The question then would be how might the qualitative values be defined and inferred from the quantitative specifications of fluctuations in the learning of feedback or feedforward GTMs. One way to address this issue is to define them in specific domains for specific properties in terms of a threshold for tolerance limits. In order for the threshold to be not dependent on the specific values, it can be specified as a percentage of fluctuation in the average value of the property. For instance, a threshold may be defined as follows: if the fluctuation in a property value is $\leq 5\%$ of the average value of the property, then it may be considered qualitatively *small*; otherwise, *large*.⁴

Similar to the scenario of learning cascading mechanism, IDEAL forms the initial hypothesis for the feedback mechanism from the two design examples. Suppose that IDEAL's analogue memory contains the design of a simple amplifier as illustrated in Figure 7.9(a, b). The structure of the device is shown schematically, its function as the pair of initial and final states of the behavior (indicated by GIVEN and MAKES in the Figure 7.9(b)), and the behavior itself as the sequence of states and transitions that explains how the structure achieves the function. The pair of states indicated by GIVEN and MAKES in Figure 7.9(b) shows the function "Amplify Electricity" of the simple amplifier. Informally, the function specifies that the amplifier takes as input electricity with a voltage of V_{in} volts (i.e., 1) at i/p and gives as output electricity with a voltage of V_{out} volts (i.e., 100 ± 20 where 100 is the average value and 20 is the fluctuation around the average value) at o/p . Figure 7.9(b) shows the causal behavior that explains how electricity applied at the input location i/p of the simple amplifier is transformed at the output location o/p . Note that the output of op-amp (operational amplifier, one of the components of the device) is dependent on the open loop gain (A_{Vo} , a device parameter) of the op-amp and is typically very high (ideally ∞) and unstable.

Let us now consider the scenario where IDEAL is presented with a problem of designing an electronic circuit. For instance, the function specifies that the desired output is electricity with a voltage value, V'_{out} volts (i.e., 100 ± 3 where 100 is the average value and 3 is the fluctuation allowed around the average value) given an input electricity of 1 volt. See MAKES and GIVEN states in Figure 7.9(d).⁵ IDEAL retrieves the design of the simple amplifier (Figure 7.9(a, b)) because the given functional specification is similar to the function of the simple amplifier. But the difference is that the output fluctuations in the retrieved design are more than those allowed

⁴In the current version of IDEAL, we use such a characterization.

⁵This is basically the problem of designing a device whose output is controlled and does not fluctuate much. A typical solution in electronics is to use an op-amp with feedback control. An op-amp is always used with feedback, whether it be in inverting or non-inverting configurations (Sedra and Smith, 1991). In the inverting configuration the input is given at the negative terminal of op-amp, and in the non-inverting configuration it is at the positive terminal.

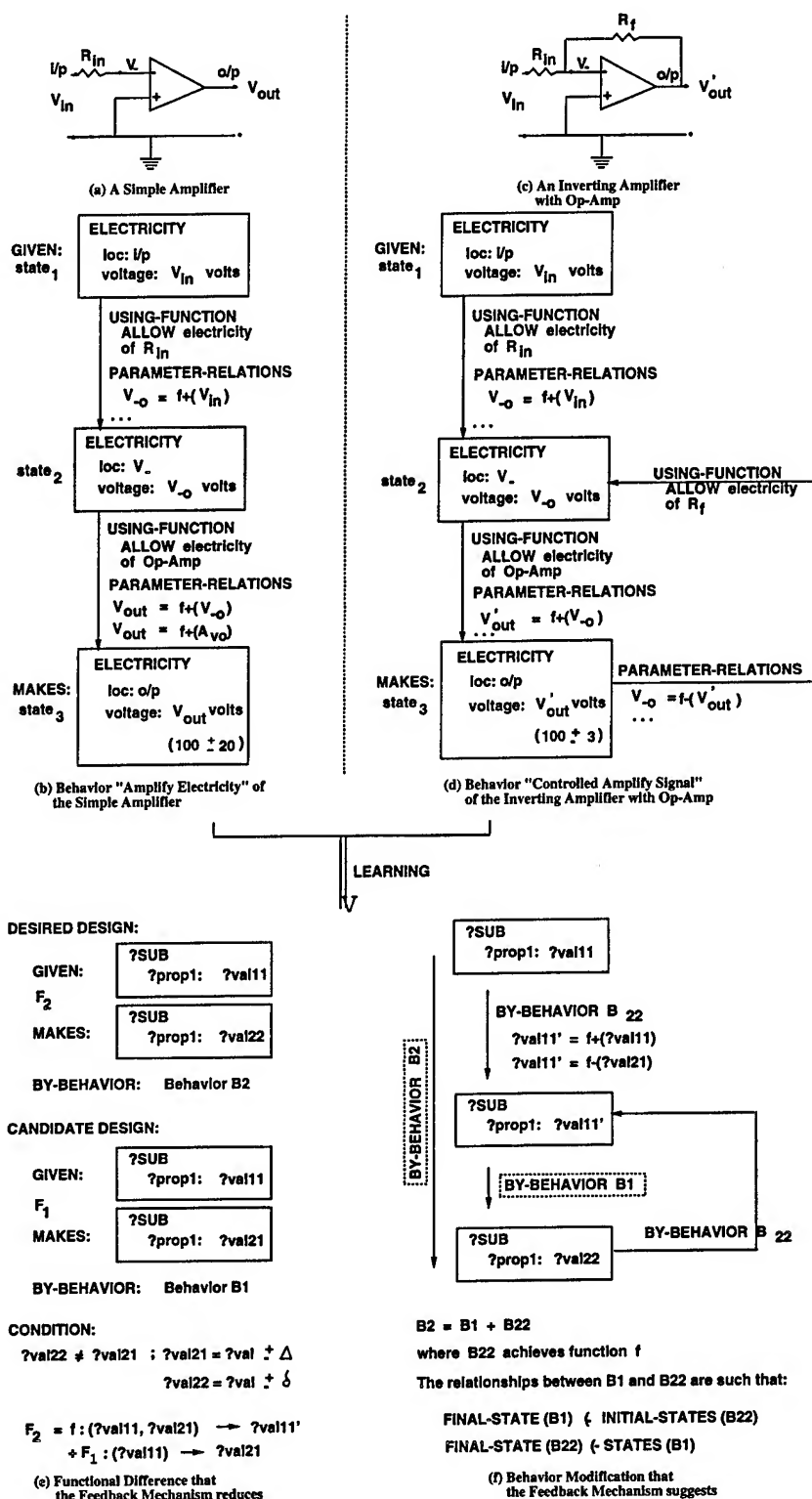


Figure 7.9: Learning of Feedback Mechanism

in the desired. The fluctuations in the output of a device can in general arise due to several reasons, for instance, due to fluctuations in the input of the device or due to unstable device parameters. In the case of the design of a simple amplifier with op-amp, for example, the fluctuations in the output voltage could be due to the device parameter, open-loop gain, A_{V_o} of the op-amp.

Suppose that IDEAL only has simple strategies such as replacing a component in a past design or substituting a new substance for one in a past design to deliver new functions. In the current scenario, given the model of the simpler amplifier as shown in Figure 7.9(b), IDEAL cannot localize the functional difference (i.e., more fluctuations than can be allowed) to any subset of components in the design because the model does not indicate any source of fluctuations. In that case, any component in the design could be a potential candidate for modification. Suppose that op-amp is chosen. Then IDEAL would only suggest that the op-amp needs to be replaced with another one that has a different A_{V_o} because that is the parameter of op-amp on which its output depends. But such replacement is not feasible in general because op-amps with any arbitrary A_{V_o} are not available! Even if the op-amp can be replaced, doing so will not satisfy the constraint on the output fluctuation. Therefore, IDEAL fails to modify the retrieved design to generate a design for achieving the new function.

Now the question is whether and how IDEAL can learn a model of the feedback mechanism if it is given the correct design for the current problem. When IDEAL thus fails to solve a problem due to its knowledge conditions, the additional constraint specified (i.e., the output fluctuations to be in certain limits), and due to the fact that some components are not available with arbitrary parameters, it has an opportunity to learn. Then, if an oracle presents the correct design that both delivers the desired function and satisfies the additional constraint (the schematic of the structure of the new device is shown in Figure 7.9(c)) and the SBF model of the new device (shown in Figure 7.9(d)), IDEAL can form the initial hypothesis for a model of the feedback mechanism.

Given the SBF models of the retrieved design (i.e., the simple amplifier) and the desired design, IDEAL compares them state-by-state and transition-by-transition along all possible dimensions in their SBF representation in order to identify the regularity between them. That is, it performs a differential diagnosis on the two models by which it determines (1) if there is a behavior segment in the new SBF model that matches with the retrieved SBF model, and (2) if so, what additional segments there are in the new SBF model and how they are related to the matching segment. Once IDEAL determines those relationships, it abstracts over the specific substances, properties and values in the relationships and forms a GTM that encapsulates the abstracted functional differences and the abstracted behavioral relationships. We will now focus on the learning of the feedback mechanism.

Recall from Section 7.2.3 that our model-based learning method involves first traversing the two focused behaviors and comparing them for similarity. When the behavior of the source design analogue (B_1) matches with (or is similar to) some segment in the new device behavior

(B_2), then there is an opportunity for IDEAL to learn a generic mechanism that specifies how to modify a behavior like B_1 to get a behavior like B_2 that achieves the function like F_2 . Also, suppose that $B_2 = B_{21} + B_{22}$, a composition of two behavior segments, and that B_1 matches with B_{21} .

In the current problem-solving scenario, applying the model-based learning method (described earlier in Section 7.2.3), IDEAL finds that the behavior segment ($state_1 \rightarrow state_2 \rightarrow state_3$ in Figure 7.9(d), excluding the transition from $state_3$ to $state_2$) in B_2 , the behavior of the inverting amplifier, matches with B_1 , the behavior of the simple amplifier ($state_1 \rightarrow state_2 \rightarrow state_3$ in Figure 7.9(b)). Continuing to traverse the additional segments in B_2 , it discovers that there is a cycle (or loop) in B_2 and picks out the relationships between the matching segment B_{21} (i.e., $state_1 \rightarrow state_2 \rightarrow state_3$) and the succeeding behavior segment (i.e., $(state_3, state_1) \rightarrow state_2$ which constitutes B_{22}) in B_2 . The functional abstraction over this segment is f and it becomes part of the decomposition of F_2 . Since in the specific behavior of the new design there are no additional states besides those in the matching segment that are not already taken into account in B_{22} , there is no subfunction g in the decomposition of F_2 . IDEAL then abstracts over the specific substances, properties and values, and the relationships to form an initial hypothesis for a generic mechanism, which is the mechanism of feedback. In order for it to abstract over the quantitative specifications of fluctuations, IDEAL needs to have a characterization of what ranges of numerical values constitute different qualitative values such as large and small. It uses the specific characterization we described in the beginning of this section. That is, it computes the percentage of fluctuation with respect to the average value and considers the fluctuation to be *small* if the percentage is $\leq 5\%$ or *large* otherwise. In general, the 5% threshold will not work for all devices or all domains; in IDEAL we use it only as a heuristic. Based on this, the fluctuation in the retrieved design of simple amplifier is abstracted to *large* and that in the new design to *small*. The model of the hypothesized feedback mechanism and its index are shown in Figure 7.9(e, f).

Note that the feedback mechanism IDEAL learned in the current scenario (Figure 7.9(e, f)) is only a partial model of the feedback mechanism because it assumes that the controlling or feedback substance is same as the controlled or output substance (?Sub)), which is not true in general.⁶ A more complete model of the feedback mechanism as illustrated in Figure 7.10 needs to distinguish between the feedback substance (?Sub_C) and the controlled substance (?Sub) as well as consider the more general decomposition of F_2 in terms of f , F_1 , and g . When the feedback substance and the controlled substance are different, the subfunction g would involve sensing the fluctuations in a property value of the controlled substance.

Note also that the feedback mechanism IDEAL had learned does not capture the subtleties of open loop feedback and closed loop feedback. Even Figure 7.10 shows only a model of closed loop feedback. In order to learn those distinctions, however, IDEAL requires more design experiences

⁶In fact, IDEAL does not even recognize that the feedback and controlled substances could be different because the current design experiences do not indicate that.

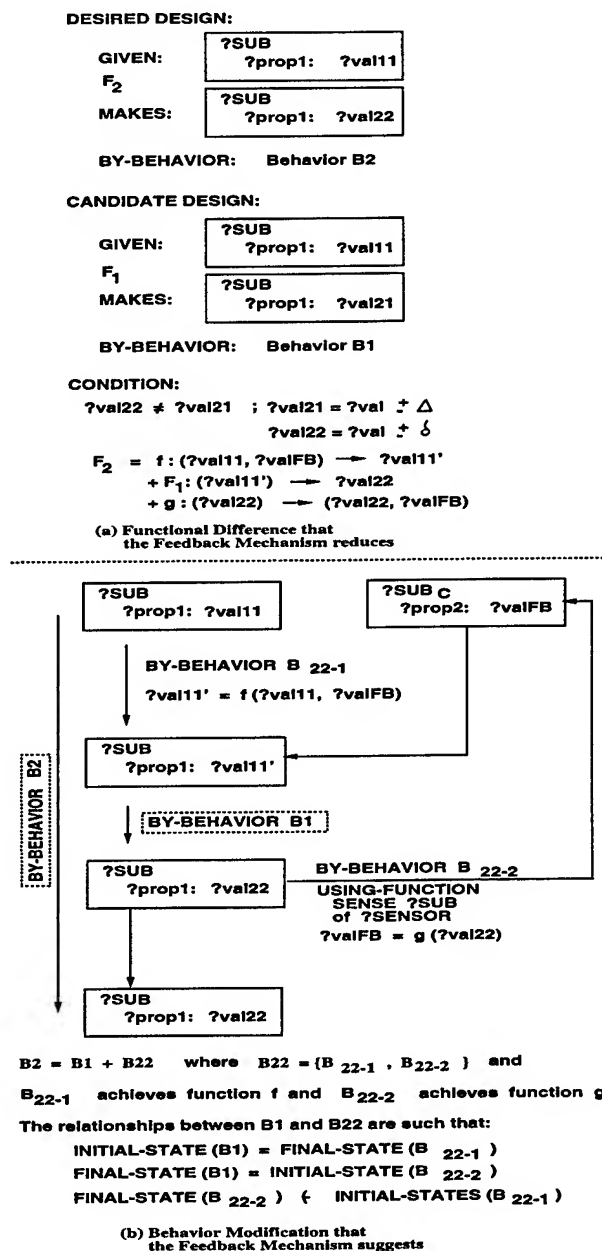


Figure 7.10: A Complete SBF Model of the Feedback Mechanism

in which the substances fed back are different and the points in the device topology to where they are fed back are different. Thus acquiring a complete model of the feedback mechanism (or, in other words, all the different types of feedback mechanism) may involve solving a number of design problems incrementally and revising the hypothesized mechanism.

7.3 Learning of GPPs

Now, let us consider the task of learning a different type of design pattern. Informally, the learning task here is: given a design analogue, form a GPP underlying the causal mechanism involved in the behaviors of the specific device model associated with the given design experience. A formal characterization of the learning task is shown in Figure 7.11.

Input:	<ul style="list-style-type: none"> • Design Analogue [consisting of design problem (i.e., function), solution (i.e., structure), and explanation (i.e., SBF model)]. e.g., design of sulfuric acid cooler.
Output:	<ul style="list-style-type: none"> • Generic physical processes (represented in SBF language). e.g., the GPP of Heat Flow and the Heat Exchange.
Method:	<ul style="list-style-type: none"> • Model-based abstraction with inductive biasing. e.g., function of a design determines what parts of the experience to focus on.
Knowledge:	<ul style="list-style-type: none"> • Typology of primitive functions in the domain. e.g., ALLOW, PUMP. • Typology of functions in the domain (consisting of primitive functions). e.g., substance-parameter transformation. • Substances in the domain. e.g., nitric acid, water. • Components in the domain. e.g., pipe, chamber. • Past design analogues in memory. e.g., design of nitric acid cooler.

Figure 7.11: Task of Learning Generic Physical Processes from Analogues

7.3.1 An Illustrative Learning Task: Learning of GPP of Heat Flow

In this section, we illustrate the task of learning GPPs with an example task of learning the GPP of Heat Flow from specific design analogues. The GPP of Heat Flow specifies how heat flows from a hot body to a cold body when they are brought in thermal contact.⁷ We describe how the models of GPPs can be acquired by a gradual removal of structural information (i.e., physical structure) from the SBF models of specific devices. This process of abstraction occurs while storing a design analogue for potential reuse.

Consider, for example, the situation in which IDEAL finds multiple (e.g., two) analogues to be similar in their functions while it is storing a new design in the functionally organized

⁷The physical principle of the Zeroth Law of Thermodynamics also captures the same (Fermi, 1937).

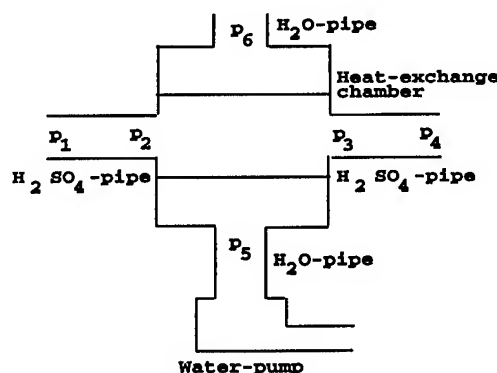
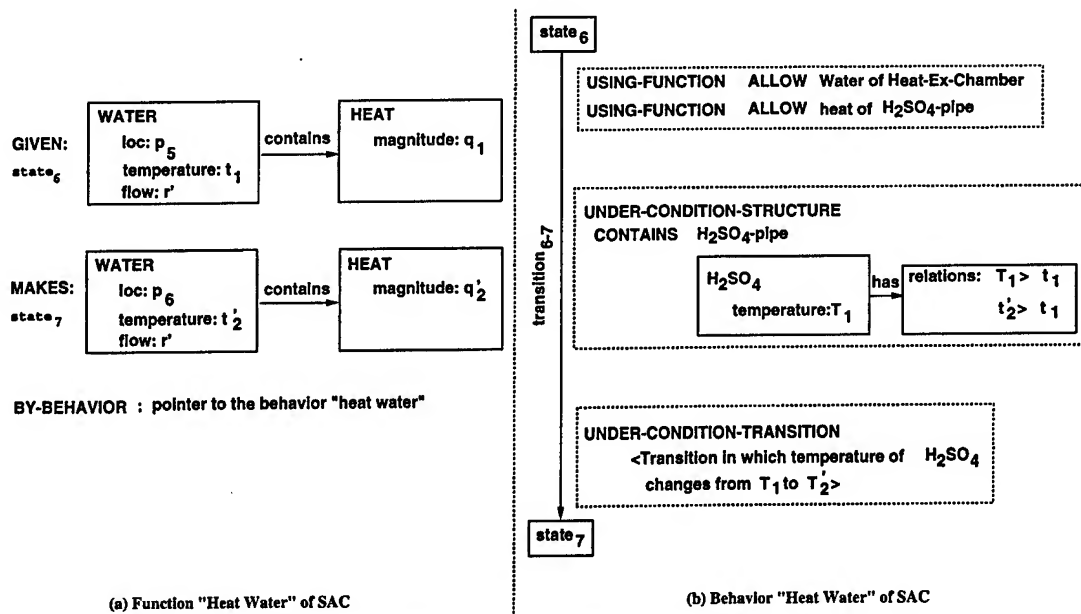


Figure 7.12: Sulfuric Acid Cooler

analogue memory. We will consider the designs of sulfuric acid cooler (Figure 7.12) and nitric acid cooler whose function and behavior are shown respectively in Figures 7.13 & 7.14 for the purpose of illustrating the task and methods. (In this illustration, we consider the “Heat Water” function of sulfuric acid cooler (SAC) and the “Cool Acid” function of nitric acid cooler (NAC).) The similarity between two functions is determined by comparing the input state and output state in them. Furthermore, similarity between two states is determined by comparing different slots in the schemas, such as **substance**, **location**, and other properties. For instance, a function F_1 is more similar to another function F_2 than it is to F_3 if the **substance** in both F_1 and F_2 is same while it is different in F_3 . For example, the function of a nitric acid cooler that cools nitric acid from T_1 to T_2 is more similar to another nitric acid cooler that cools nitric acid from T_1 to T_3 than it is to a sulfuric acid cooler that cools sulfuric acid from T_1 to T_2 . This is based on the heuristic that changing a substance altogether in a design is harder than changing a property of a substance. These similarity measures are based on those used in KRITIK for accessing cases from memory (Goel, 1992b). In addition to abstracting the functions of similar design analogues, IDEAL can also abstract the associated SBF models for use in solving problems by analogy in a different domain with the experience gained in one domain. However, IDEAL does not know *a priori* what the target “concept” will be; hence, it formulates the abstracted model as a hypothesis.

As mentioned earlier, the function of a device determines what parts of its model to abstract. If the function is a *transformation function* (e.g., substance transformation, substance-parameter transformation, substance-location transformation) then any relations in the different types of context annotating the transitions in the behavior that describe the corresponding change and the transitions themselves can be abstracted to form meaningful abstractions of behaviors. For example, since the function “heat water” of sulfuric acid cooler is to transform the **temperature** of the substance **water** from one value to another, the transition *transition₆₋₇* in Figure 7.13(b) is useful to focus on. The relations on the parameters of **temperature** describing the change can be abstracted along with the similar behavior of another cooler or heater. In addition to



Note: All locations are with reference to components in this design.
All labels for states and transitions are local to this design.

Figure 7.13: Function and Behavior of Sulfuric Acid Cooler

the parametric relations, other aspects of the context, such as conditions on substance and conditions on structural relations that involve the parameter being transformed, also form an important part of the content to be abstracted.

After identifying what parts of the specific models to focus on, the issue is to determine what kinds of changes along a dimension are meaningful for abstraction. In other words, the issue is what similarities between the two models (in the focused segments of the behaviors) are retained, as they are, in the abstraction and what differences are abstracted. The same kind of similarity metrics as those for comparing functions are used for this purpose as well, because a focused segment of behavior includes a sequence of states and state-transitions. However, in addition to comparing states, the annotations on the transitions are also compared as guided by the functions (explained above). Since abstractions tend to deal with more qualitative parameters than specializations, we consider *positive* changes (i.e., increase) and *negative* changes (i.e., decrease) in the parameter of the chosen property for abstraction. The changes across different models under consideration suggest the level of abstraction. Since SBF models specify different kinds of structural information (e.g., locations, structural relations, components etc.), successive removal of each kind leads to the formation of models at different levels of abstraction. By removal we mean two things: (1) substitution of specific values (e.g., low and medium) by a value from a more general class of values (e.g., qualitative-value) in a value hierarchy and (2) a complete deletion of specific structural information (e.g., deleting the information that some substance *moves* from one location to another). These will become clearer from the example

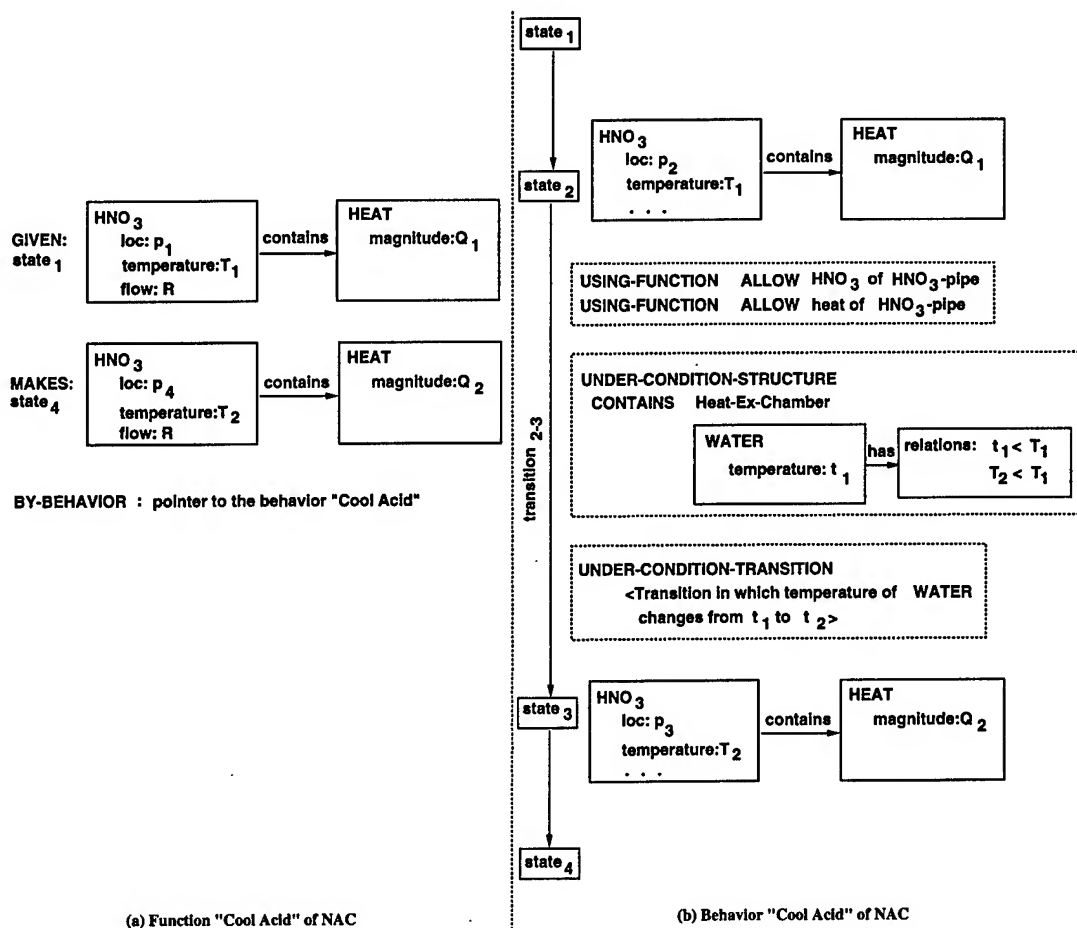


Figure 7.14: Function and Behavior of Nitric Acid Cooler

illustrated below.

7.3.1.1 Learning to Different Levels of Abstraction: Heat Exchange and Heat Flow GPPs

Since some functions such as that of a sulfuric acid cooler can be classified in multiple ways, multiple subtasks of abstraction can be performed—abstraction over parameter changes and abstraction over changes in location. Depending on which abstraction is performed on given experiences, different types of abstract models will be formed. However, in some cases, both might be applicable; in such a case of multiple subtasks, abstraction occurs to multiple levels. IDEAL applies both methods, when applicable, in a specific order, that is, it abstracts over parameter changes prior to changes in location. Models at intermediate levels of abstraction are models of prototypical devices (similar to design prototypes (Gero, 1990)) such as the model of

a heat exchanger that is applicable to both coolers and heaters. Models at still higher levels of abstraction are such as the model of a physical principle “the zeroth law of thermodynamics” or the physical process “heat flow.”

7.3.2 The Model-Based Learning Methods

Consider the design of a sulfuric acid cooler (Figure 7.12) and its function of heating water for the purpose of illustrating the methods. The type of this function (i.e., substance-parameter transformation as well as substance-location transformation) suggests two methods for abstraction: (1) abstraction over substance-parameter transformation (Figure 7.15) and (2) abstraction over substance-location transformation (Figure 7.16). The transitions *transition*₂₋₃ in the behavior “cool acid” of NAC (Figure 7.14(b)) and *transition*₆₋₇ in the behavior “heat water” of SAC (Figure 7.13(b)) are selected for abstraction because they transform parameters of the substance temperature and the substance location.

The application of the method shown in Figure 7.15 to these two behaviors results in the description of an abstracted model as shown in Figure 7.17, which is the SBF model of the GPP of Heat Exchange (i.e., the model of a heat exchanger which is a prototypical device). Note

Input: • E_1 , the new design experience.
 • E_2 , a design experience found to be similar to E_1 under the same node in memory.

Output: • Abstracted model from E_1 and E_2 .

Procedure:

if (function of E_1 is substance-parameter-transformation)

then

begin

 (1) Get transitions, TR_1 and TR_2 , corresponding to the transformed parameter in E_1 and E_2 respectively.

 (2) Compare the change in parameters in TR_1 and TR_2 qualitatively.

if (direction of change is same in TR_1 and TR_2)

then abstract over “range” of the parameters;

else abstract over the direction of change;

 (3) Modify other context in TR_1 and TR_2 that specifies this parameter. That is,

if (any “inequalities” exist on the parameter-relations)

then abstract the inequalities to conditional inequalities;

 (4) Propagate this abstraction to other dependent parameters and transitions, and then repeat step (3) until all the context is abstracted.

 (5) Store the abstracted model from E_1 and E_2 .

end.

Figure 7.15: A model-based method for abstracting over parameter transformation

that the structural information in the behaviors of SAC and NAC is abstracted and so are the

parametric relations in the corresponding transitions (Figure 7.17). For instance, the specific components H_2SO_4 -pipe and HNO_3 -pipe that achieve the function “allow heat” are abstracted to the abstract component pipe achieving the same function, which is prototypical of a heat exchanger.

IDEAL’s knowledge of components that H_2SO_4 -pipe and HNO_3 -pipe belong to the class of *pipes* helps in doing this abstraction. Also, the parametric relations in Figure 7.17 cover both possibilities, that is, *increase* and *decrease* in the substance temperature, unlike those in the behavior of either SAC or NAC alone. This is essential to describing the behavior of a heat exchanger. Further, the abstractions are propagated to the behaviors of those substances on which the transitions depend, which is indicated by UNDER-CONDITION-TRANSITION in the Figures 7.13(b) & 7.14(b). That is, in step 4 of the method (Figure 7.15), for instance, the abstractions performed on the behavior segment (say, “heat water” of sulfuric acid cooler) are propagated to the dependent transition (i.e., “cool acid” of sulfuric acid cooler) which results in the abstracted segment “cool substance” shown in Figure 7.17.

Input: • E_1 , the new design experience or newly abstracted experience.
 • E_2 , a design experience (perhaps abstracted before) found to be similar to E_1 ,
 if any, under the same node in memory.

Output: • Abstracted model from E_1 (and E_2).

Procedure:
 if (function of E_1 is substance-location-transformation)
 then
 begin
 (1) Get transitions, TR_1 and TR_2 , corresponding to the location in E_1 and E_2 respectively.
 (2) Compare the causal context that involves location in TR_1 and TR_2 .
 if (causal context is similar in TR_1 and TR_2)
 then abstract/variablize locations;
 else abstract over the associated structural elements;
 (3) Modify other context that involves locations and associated structural information.
 That is,
 if (any structural conditions exist in TR_1 and TR_2 and they are similar)
 then remove the structural conditions;
 else check for similarity at a more abstract level of components involved;
 (4) Propagate this abstraction to other dependent parameters and transitions,
 and then repeat step (3) until all the context is abstracted.
 (5) Store the abstracted model from E_1 and E_2 .
 end.

Figure 7.16: A model-based method for abstracting over location transformation

The application of the method shown in Figure 7.16 to the result of applying the first method, that is, to the model of the heat exchanger, leads to the formation of an even further abstracted

model as shown in Figure 7.18. This is a partial description of the generic physical process “heat flow” or the principle that we call the zeroth law of thermodynamics.⁸ However, the system, conforming to the classical “term problem” in learning, does not realize that this is *the* zeroth law of thermodynamics nor does it realize that this is a partial description of the process “heat flow,” but rather considers it simply as an abstract model possibly applicable to a wider class of devices. Again, note that the structural information in the behavior of heat exchanger is further abstracted in the behavior of heat flow. For instance, the component pipe that achieves the

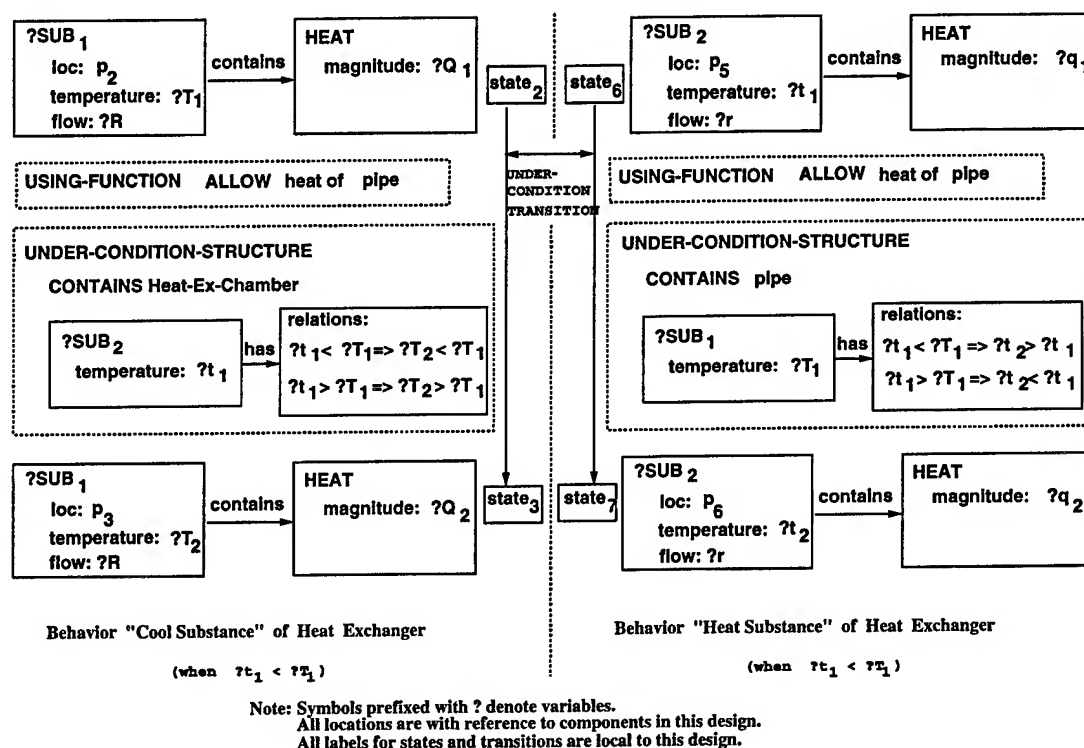


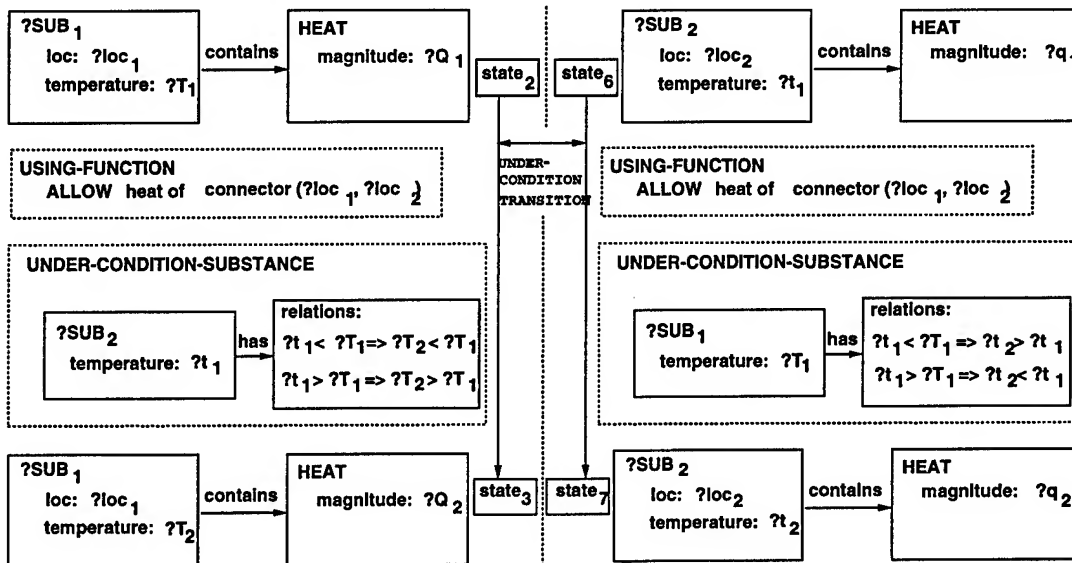
Figure 7.17: Behavior of a Heat Exchanger (i.e., the GPP of Heat Exchange) abstracted from SAC and NAC

function “allow heat” is abstracted to an abstract component connector achieving the same function.

In addition to the result of abstraction over structure, the abstracted parametric relations in Figure 7.18 that cover both *increase* and *decrease* in the substance temperature are also crucial to representing the heat flow GPP. These relations are represented as conditions on substance properties indicated by the annotation UNDER-CONDITION-SUBSTANCE in Figure 7.18 because the structural conditions (in Figure 7.17) are removed by the application of step 3 in

⁸A complete description should also indicate that heat continues to flow from a hot body to a cold body only until an equilibrium temperature is reached.

Figure 7.16. Again, step 4 in the method shown in Figure 7.16 leads to the propagation of abstractions performed in one behavioral segment to the dependent ones.



Note: Symbols prefixed with ? denote variables.
All locations are with reference to components in this design.
All labels for states and transitions are local to this design.

Figure 7.18: The Zeroth Law of Thermodynamics (i.e., the GPP of Heat Flow)

CHAPTER VIII

LEARNING DEVICE MODELS

In the previous chapter, we considered four different interaction conditions (3 under problem-solving failure, and 1 when there are no problem-solving failures) for knowledge acquisition. In brief, the three conditions under problem-solving failure were (1) oracle provides all the three constituents of the target design analogue, i.e., the function, structure and behavior for the new design; (2) oracle provides only the function and structure for the new design; and (3) oracle provides only the function and the localized substructure in the context of the failure (i.e., a solution for the specific, local adaptation goal). The fourth condition was when there was no problem-solving failure, i.e., the MBA process generates a design by exploring with alternative strategies, perhaps taking more modification steps than necessary.

In all the four conditions, an SBF model of the new device is acquired. The learning of an SBF model of the device is important because the SBF model, in turn, enables several subprocesses in MBA including learning of design patterns, learning of indices, and transfer and modification of this device in future problem solving. In the first condition above, since the oracle directly gives the SBF model of the new device the acquisition is trivial. But in the other conditions, acquiring a new device model can be computationally hard. The fundamental issue common to learning in all those conditions is how to derive the internal causal behaviors of the new device given its structure. This task is hard because in some domains structural elements may have multiple behaviors and selecting the appropriate behavior(s) for each element in the given structure and composing them to generate the behaviors for the entire structure can be very complex. It is more so when the given structure is complex. Therefore, the computational issue is how to control the inferences needed in achieving this task.

The models of devices can be generally acquired in a number of ways: acquisition from a teacher, acquisition from natural language descriptions of devices and their behaviors (Peterson et al., 1994), acquisition by composition of primitive structural elements (i.e., consolidation (Bylander, 1991)), and acquisition by revision of models of similar devices (Goel, 1991b). We introduce two new methods for the acquisition of device models. In our theory, new device models may be acquired by the following three methods: (1) revision of models of known devices as in KRITIK (Goel, 1991b), (2) a combination of model revision and composition of behaviors of primitive structural elements, and (3) instantiation of design patterns in the models of known devices. All three methods involve revising the models of known, similar devices. By avoiding the need for composition of behaviors of each primitive element in the structure, they control

the inferences.

IDEAL uses methods (1) & (2) for acquiring a new SBF model under the interaction conditions (2) & (3). But IDEAL uses all three methods under condition (4). Since we already described method (3) in Chapters 5 & 6, we focus here on methods (1) & (2). In particular, we will describe how IDEAL acquires a new SBF model using method (1) under condition (4) and using method (2) under conditions (2) & (3).

8.1 Learning by Model Revision and Primitive-Behavior Composition

IDEAL uses this method in two of the interaction conditions under problem-solving failure: when it receives only the desired design (i.e., the structure) and when it receives only the solution to the specific adaptation goal. This new method is necessary because of the following reasons. Although the method of behavior composition can be used for deriving behaviors from most structures, it is computationally expensive and hence not desirable. Although the method of model revision addresses the complexity issue, it alone is not sufficient for solving all classes of structure-to-behavior generation problems. For instance, when the given structure has new components and the known devices do not have them, then revising the model of a known device requires composition of the behaviors of the new components. Therefore, we developed this new method that combines model revision and composition of primitive behaviors.

8.1.1 When Oracle presents only the Desired Design upon Problem-Solving Failure

Since IDEAL uses a model-based method that requires the SBF models of the source and target design analogues for it to learn any abstractions, under this condition of interaction, it requires the additional inference step of deriving the behavior for the given structure of the new design. Therefore, given the structure and the function (available as part of the problem specification) of the new desired design, IDEAL first derives the internal causal behaviors of the given structure in order to comprehend how the given structure achieves the desired function. It uses the method of model-revision and primitive-behavior composition for this task; that is, it revises the behavior of the retrieved design analogue by mapping the components in the given structure onto those in the structure of the retrieved design and by composing the behaviors of any additional components in the new structure with the behavior of the retrieved analogue.

Once IDEAL comprehends the functioning of the given structure in terms of the internal causal behaviors of the structure, it can learn design patterns as described in the previous chapter. We have tested IDEAL for its learning of SBF models under these knowledge conditions in the domains of electric circuits, heat exchangers, and electronic circuits (with operational amplifiers) in the context of learning cascading and feedback mechanisms. We will now illustrate its learning of the SBF model for EC3 (the 3-volt Electric Circuit that produces light; Figure 7.3)

from that of the design, EC1.5 (the 1.5-volt Electric Circuit that produces light; Figure 7.2) under the current knowledge conditions.

```

STRUCTURE    Circuit EC3
COMPONENTS: (Battery1, Battery2, Switch, Bulb)

    STRUCTURE    Battery1
    RELATIONS: (SERIALLY-CONNECTED
                  Switch Battery2)
                  . . .
    PARAMETERS: (voltage 1.5volts)
    FUNCTIONS: (ALLOW electricity)
                  (PUMP electricity)
    CONNECTING-POINTS: (T2' T2)

    STRUCTURE    Battery2
    RELATIONS: (SERIALLY-CONNECTED
                  Battery1 Bulb)
                  . . .
    PARAMETERS: (voltage 1.5volts)
    FUNCTIONS: (ALLOW electricity)
                  (PUMP electricity)
    CONNECTING-POINTS: (T3 T2')

    STRUCTURE    Switch
    RELATIONS: (SERIALLY-CONNECTED
                  Battery1 Bulb)
    MODES: (open closed)
    . . .

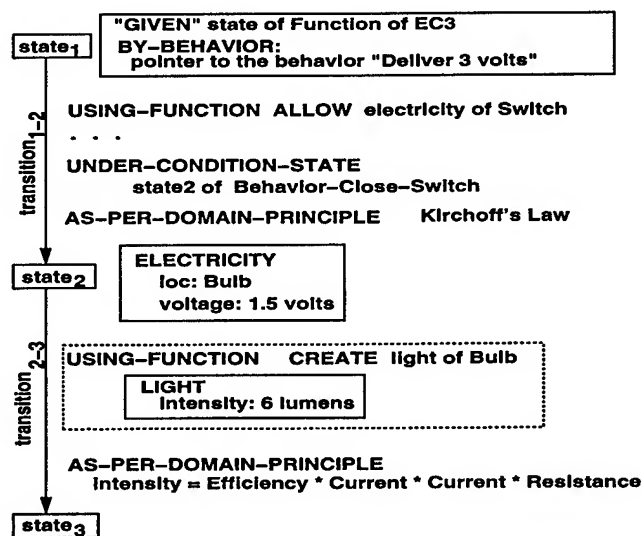
    STRUCTURE    Bulb
    RELATIONS: (SERIALLY-CONNECTED
                  Switch Battery2)
    PARAMETERS: (resistance 5 ohms)
                  (efficiency 6.66)
                  (wattage 20)
    FUNCTIONS: (CREATE light)
                  (ALLOW electricity)
    CONNECTING-POINTS: (T3 T4)

```

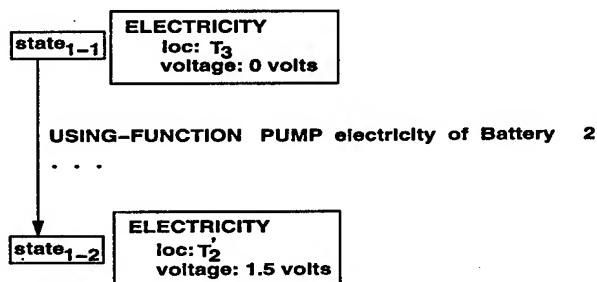
Figure 8.1: Structure of the 3-volt Electric Circuit in Schema Form

Recall from Chapter 7 (Section 7.2.1) the situation in which IDEAL fails to generate a design for the problem of EC3 by modifying the retrieved design analogue EC1.5. Suppose that IDEAL is now given the structure of a correct design for achieving the function of EC3 as shown in Figure 8.1. As a first step, IDEAL copies the behavior of EC1.5 and modifies each unitary behavior segment (i.e., single state-transition-state) by mapping a similar component in the new structure onto the one in the behavior. For mapping these components, it uses the structural context (i.e., what other components to which the particular component is connected and how it is connected) and the functions of the particular components. For instance, the Bulb in EC1.5 is mapped only onto the Bulb in EC3 (and not onto any other component) because the structural context (i.e., a serial connection with a battery at a particular end and a serial connection with

switch at a particular end) and functions of Bulb in EC1.5 (in general, all its functions) are respectively similar to the structural context and functions of Bulb in EC3. These principles of mapping are similar to the *systematicity principle* of the structure-mapping theory of (Gentner, 1983). These principles however do not guarantee a single, unique mapping between the structure of the retrieved design and the new design. For instance, either of the batteries in EC3 can map onto the battery in EC1.5, but both mappings are equivalent in this particular case because of symmetry in the connections. In general, IDEAL can work with any of these multiple valid mappings, but it chooses one randomly and proceeds with it. Hence, let us suppose that it maps the battery in EC1.5 onto Battery2 in EC3. The incomplete and incoherent behavior for the new structure before adding the behavior of the additional component (i.e., Battery1 in EC3) will be as shown in Figure 8.2.



(a) Behavior "Produce Light" of EC3



(b) Behavior "Deliver 3 volts" of Battery Structure

Note: All locations are with reference to components in this design. All labels for states and transitions are also local to this design.

Figure 8.2: Incomplete & Incoherent Behaviors of the 3-volt Electric Circuit in the middle of the process of generating behavior from structure

In order to revise the incomplete behavior to include the behaviors of any additional components in the new structure, for each additional component IDEAL first needs to determine *where* to compose the component behavior. That is, it needs to determine *at which point* in the incomplete behavior it should add the new component behavior. This task requires that the states in the behavior be specified at the *locations* between components (i.e., the structural points at which components are connected). So, for instance, IDEAL finds that the behavior of Battery1 in EC3 needs to be added after the state at location T_2' in the incomplete behavior of EC3 (Figure 8.2). For each component, its behavior is specified as a transition between two states, the states at the input end and output end of the component. Based on the points at which the component behavior will be added in the incomplete behavior of the given structure and the input and output ends of the component, first the input and output states of the component behavior are generated. Then one of the functions of the component that best describes the transformation between those input and output states is selected to describe the transition between them. That is necessary because a component can have multiple functions, only one of which may be relevant in the current device. In the specification of components, each of its functions also specifies qualitative relations between parameters at the input end and the output end. When a function of the component is selected to describe the transition, the corresponding qualitative relations are also selected. Then those relations are modified to reflect the parameters in the input and output states of the component behavior, and added in the transition. For instance, the component behavior thus generated for Battery1 in EC3 is shown in Figure 8.3. Then this behavior of Battery1 is added after the state at location T_2' (i.e., $state_{1-2}$) in the incomplete behavior of EC3, and the parameter changes due to this insertion are propagated to the subsequent states and transitions in the incomplete behavior to generate a complete and coherent behavior for the given structure of EC3 as shown in Figure 8.4.

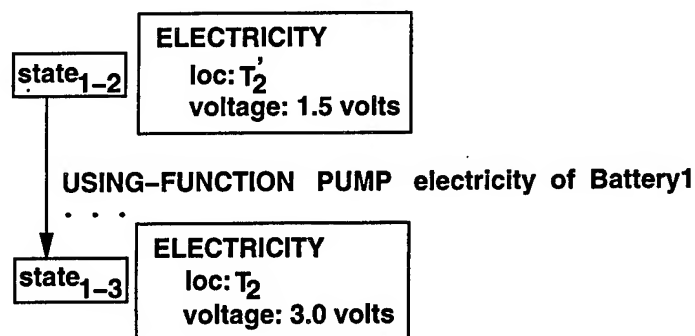
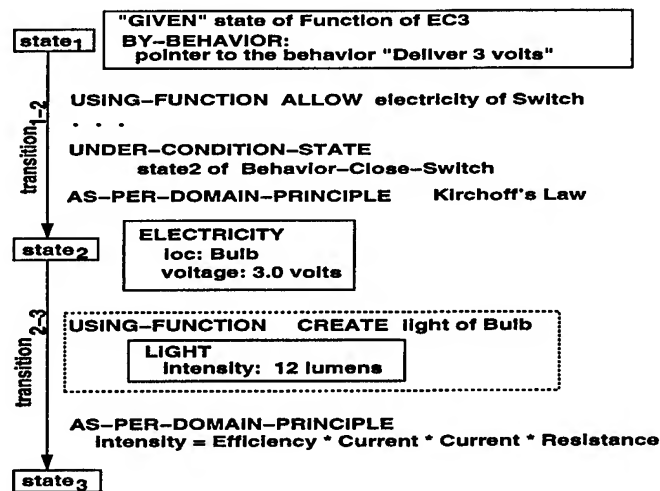
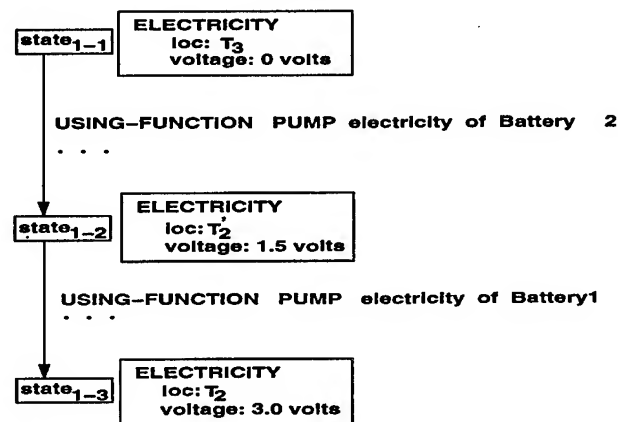


Figure 8.3: Behavior of Battery1 in the context of the 3-volt Electric Circuit



(a) Behavior "Produce Light" of EC3



(b) Behavior "Deliver 3 volts" of Battery Structure

Note: All locations are with reference to components in this design. All labels for states and transitions are also local to this design.

Figure 8.4: Complete Behaviors of the 3-volt Electric Circuit generated from the given structure

8.1.2 When Oracle presents only the Solution to the Specific Adaptation Goal upon Problem-Solving Failure

This section describes our exploration of IDEAL's learning under a different situation. The input knowledge conditions in this situation are such that when IDEAL fails to generate a design for a new problem it is given only the localized structure for a correct design. Recall that the computational process of MBA (Chapter 2) that IDEAL uses is such that it localizes, whenever possible, its modifications to a retrieved design analogue while solving a new design problem. Therefore, when it fails to generate a design, it has the knowledge of the localized structure in the retrieved design that it could not modify. In such a situation, if IDEAL is given the

correct localized structure,¹ then in order to learn any generic mechanism it needs to do two additional inference steps. The first is to revise the structure of the retrieved design to include the given new localized structure, and the second is to revise the behavior of the retrieved design to generate the internal causal behaviors for the new structure.

Once IDEAL generates the structure for the new design problem, it can generate its internal causal behaviors as described in the previous section, and then learn design patterns as described in the previous chapter. We have tested IDEAL for its learning of SBF models under these knowledge conditions in the domains of electric circuits, heat exchangers, and electronic circuits (with operational amplifiers) in the context of learning cascading and feedback mechanisms. Since we have described in the previous section IDEAL's process for going from the structure of the new design to form its SBF model, we will now focus on the generation of the structure from a given localized structure. We will illustrate its process with the designs of EC1.5 and EC3 from the overall context of learning cascading mechanism under the current knowledge conditions.

Recall from Chapter 7 (Section 7.2.1) that when IDEAL failed to generate a design for the problem of EC3 by modifying the design of EC1.5, the localized structure it was trying to modify was the Battery. Now let us suppose that IDEAL is given the new localized structure (i.e., just the part of the structure of EC3 that contains Battery1 and Battery2) to replace the Battery in EC1.5. Since IDEAL knows which localized structure in the retrieved design to replace and what is the new localized structure, it can very easily revise the structure of the retrieved design to reflect the new localized structure as its part and correspondingly revise all the structural relations that are involved. That is possible precisely because its learning is situated in its problem-solving context. However, one important, necessary inference it needs to draw before revising the structure of the retrieved design is to determine the mappings between the *ends* (the input and output ends) of the old and new localized structure (i.e., the connecting-points at which these structures are connected to the remaining structural elements in the design). The SBF language for structure enables IDEAL to perform this mapping consistently. That is, IDEAL maps the input end of the old structure onto the input end of the new structure, and so it does for the output ends also.² This is to make sure that the direction of flow of substance(s) in the new structure is consistent with that in the old structure. Figure 8.5 illustrates the structure of the design of EC1.5, the new sub-structure that generates 3 volts, and the structure of the design of EC3 that IDEAL generates from the former two. Once the structure of the new design is available, IDEAL can generate its internal causal behaviors as described in the previous section.

¹the overall desired function is known as it is part of the problem specification and the function of the localized structure is also known as the problem-solving context provides that.

²In general, it is more difficult and not clear how to infer the mappings when there are multiple input ends (for instance, there are two input ends for an op-amp) or multiple output ends. That remains as an open issue.

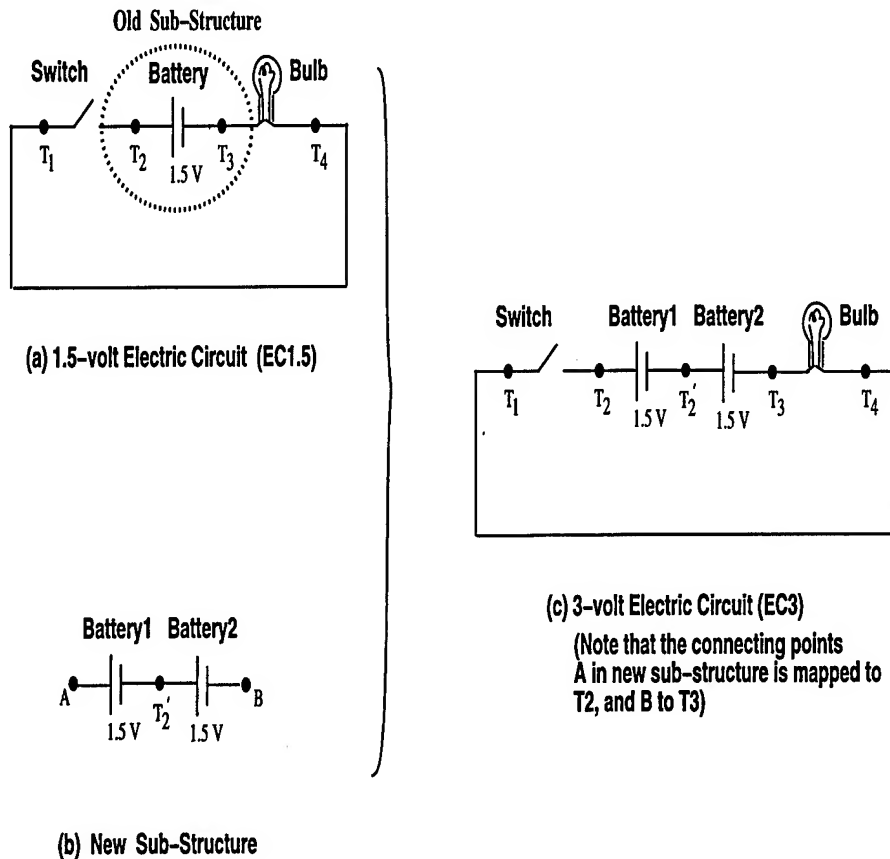


Figure 8.5: Illustration of Generating the Structure of EC3 from EC1.5 and the New Sub-Structure (all structures shown only diagrammatically)

8.2 Learning by Model Revision

IDEAL uses this method for generating the behavior of a new design when the modifications to the retrieved design are simple and when there are no problem-solving failures. In this section, we illustrate how it can use this method repetitively to make a complex modification to the behavior of the retrieved design.

8.2.1 When there are no Problem-Solving Failures

Until so far, we have only considered how IDEAL could learn in a failure situation (i.e., it either cannot generate a design for the given problem or it generates an incorrect design) under different interaction conditions. We will now describe how it can learn a new SBF model completely autonomously. Obviously, this task requires IDEAL to be endowed with some alternative strategies so that it can use them in solving new design problems successfully; it may have to do some exploration with this alternative source of knowledge in order to be able to solve the

new problems.³

To the extent this research has explored, there were no alternative strategies for the mechanisms of feedback, feedforward, and device composition. But, for the cascading mechanism, an alternative is component-addition strategy. Hence, the discussion here will concentrate on how the strategy of component addition and the exploration to repetitively apply it in a problem-solving context enable IDEAL to solve the design problems and generate the new SBF model in which the cascading mechanism can be learned.

The strategy of component addition suggests that a component, whose functional transformation is \leq the difference between the desired transformation and that of a candidate component, can be added (serially) to the candidate component in order to reduce the difference.⁴ Therefore, when a component whose function is same as the difference between the desired and that of a candidate in a retrieved design analogue can be found, component addition leads to generating in one step a design that achieves the new function; it will require more than one step to generate a design if the available components only achieve functions that are $<$ the difference. That is, for instance, when the desired function is to achieve 2.5 volts and a candidate component achieves 1.5 volts, component addition suggests that a component that achieves 1 volt can be added to the candidate. Similarly, when the desired function is to achieve 4.5 volts and a candidate component achieves 1.5 volts, it suggests to add in two steps two 1.5-volt components to the candidate; that is when there is no single 3-volt battery available. With the knowledge of component addition, IDEAL can solve all those problems under which it can learn the cascading mechanism.

Let us now consider how the repetitive use of the component-addition strategy in a problem-solving context leads to a design (that further enables IDEAL to learn a model of the cascading mechanism). Recall from Chapter 5 that IDEAL's adaptation process involves first revising the model of the retrieved design and only then its structure. To make the discussion concrete as well as related to the earlier discussions of learning models, we will consider the same problem of designing EC3. Furthermore, we suppose that IDEAL has retrieved the design of EC1.5. IDEAL recognizes that the component-addition strategy is applicable in this context because the functional difference between the problems of EC3 and EC1.5 matches with what the component addition can reduce and there is a component available (a battery with 1.5 volts) whose function is \leq to the functional difference to be reduced (which is 1.5 volts). Hence applying the component addition with a battery of 1.5 volts to the design of EC1.5 involves adding the battery in series with the one already in EC1.5. As per the computational process of model-based analogy, the modification suggested by the strategy is instantiated first in the behavior of the retrieved

³This implies that the alternative mechanisms provide IDEAL with the same *competence* as would any "new" generic mechanisms that may be learned with oracle interaction under failure after learning the new SBF model. But the new generic mechanisms learned in this situation may make a difference in the *performance* of IDEAL, that is, they enable it to solve similar design problems faster. Thus IDEAL's learning of GTMs in these situations is equivalent to knowledge compilation.

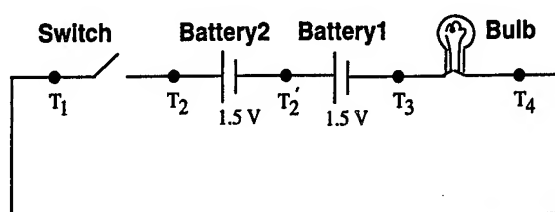
⁴along the same motivations as *means-ends analysis* suggests.

design. Then the behavior is simulated to make sure that the local modification is consistent with the overall function and that it reduces the functional difference. Finally, the structure of the retrieved design is revised. It generates the behavior of the new design by revising that of EC1.5 (Figure 7.2) with the addition of the behavior of 1.5-volt battery (Figure 8.3) after the state $state_{1-2}$. The resulting behavior is same as shown in Figure 8.4. Since in the current context, IDEAL finds by simulating the model that the addition of the 1.5 volt battery to the one in EC1.5 reduces the functional difference to *null*, the generation of a design for the problem of EC3 is considered complete.

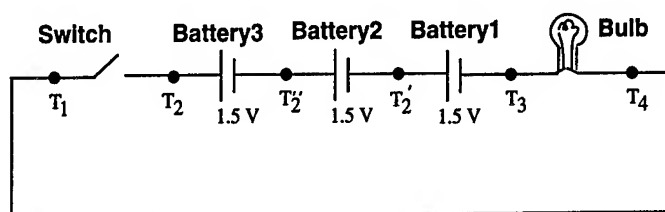
In the above example, only one application of the component-addition strategy was sufficient to complete the design problem. But in general, it may require several repetitive applications of the strategy. In order to illustrate such an exploratory process, let us consider a different design problem, namely, that of designing a 6-volt Electric Circuit (EC6). The function of the new desired design can be informally specified as producing light of intensity 24 lumens in the bulb as output when the switch is closed, given an electricity with a voltage of 6 volts in the battery as input. Similar to the earlier problems we considered, the new design problem also specifies a structural constraint that the desired design cannot have a single 6-volt battery. Furthermore, suppose that a constraint from the memory of available batteries is that there are only 1.5-volt batteries to use. Supposing that IDEAL has the design of EC1.5 in its memory of analogues, given the new problem, it can retrieve the design of EC1.5 because their functions are similar.

IDEAL recognizes that the component replacement will not work in this problem due to the structural constraint, but instead, the component addition would. The component-addition strategy is applicable because the functional difference between the problems of EC6 and EC1.5 matches with what the component addition can reduce and because there is a component available (a battery with 1.5 volts) whose function is \leq to the functional difference to be reduced. When testing the applicability of the strategy, IDEAL first looks for a component with a function exactly equal to the difference to be reduced (i.e., a battery with 4.5 volts); if such a component cannot be found, then it looks for a component that reduces the difference most (i.e., the battery with the most voltage capacity $<$ the difference to be reduced). In the current context, such a component would be a battery with only 1.5 volts because of the constraints from the memory of available components. Hence, applying the strategy with the 1.5-volt battery in the design of EC1.5 will only produce a design as shown in Figure 8.6(a). The behavior of EC1.5 (Figure 1.4) is revised by adding the behavior of 1.5-volt battery (similar to one shown in Figure 8.3) resulting in the behavior shown in Figure 8.7(a). IDEAL can recognize by simulating the behavior of this design that it does not achieve the desired function. Then, IDEAL starts to explore with the partial design thus far generated. That is, it views the partial design as a retrieved design analogue and applies the strategy of component addition to it. The second application of the strategy results in the partial design shown in Figure 8.6(b). The resulting behavior due to the second application of component addition is shown in Figure 8.7(b). Note that although the second partial design is closer to achieving the desired function than the first one, it does not

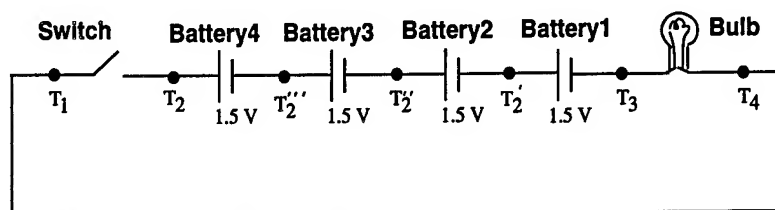
completely achieve the desired function. IDEAL thus continues to explore with the partial design until either it arrives at a design that achieves the desired function or it reaches any resource limitations (such as an upper bound on the number of repeated steps of transfer & modification). In the current context, the third application of the strategy results in the complete design for the given problem as shown in Figure 8.6(c). The behavior of the final design after revising that in the previous step is shown in Figure 8.7(c). Thus by making a simple modification to the model of a retrieved design in each step, IDEAL can make a complex modification in multiple steps that is equivalent to the one due to the instantiation of cascading mechanism in the retrieved design.



(a) Partial Design for EC6 after One Application of Component-Addition Strategy on EC1.5



(b) Partial Design for EC6 after Two Applications of Component-Addition Strategy on EC1.5



(c) Design for EC6 after Three Applications of Component-Addition Strategy on EC1.5

Figure 8.6: Evolving Design for EC6 with repeated Applications of Component-Addition Strategy on the Design of EC1.5

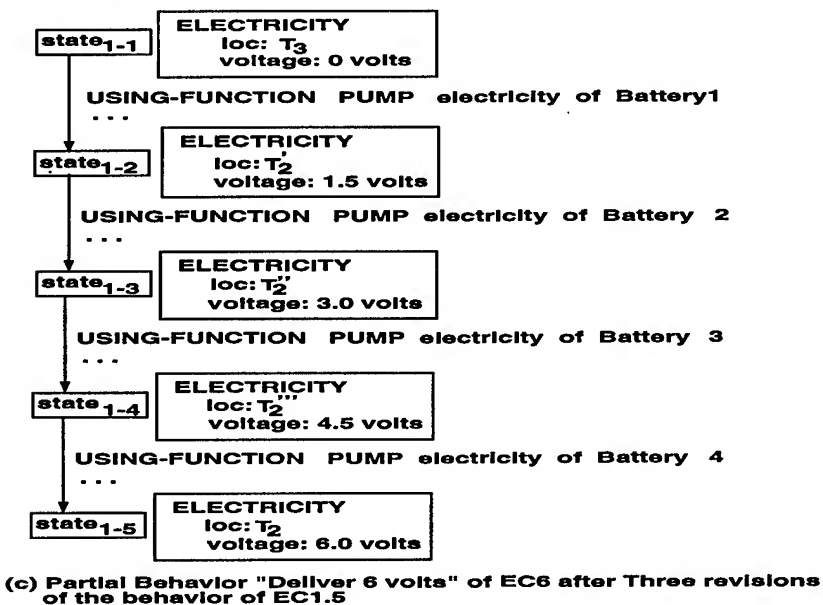
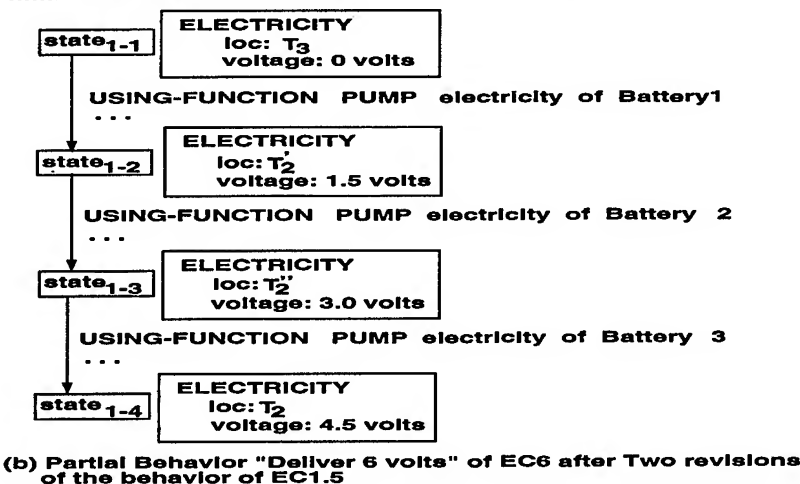
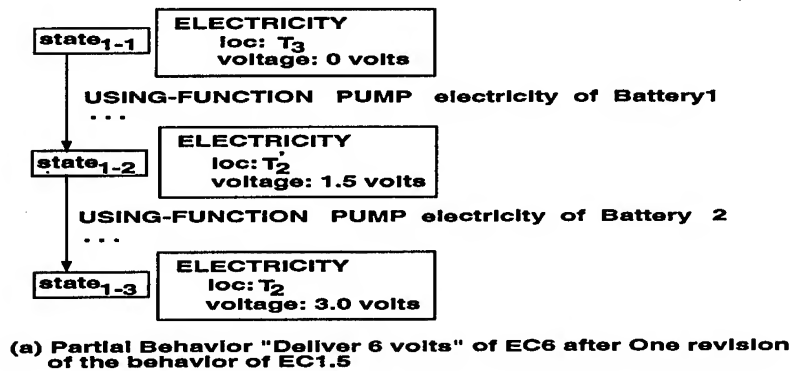


Figure 8.7: Evolving Behavior for EC6 with repeated Applications of Component-Addition Strategy on the Behavior of EC1.5

CHAPTER IX

EVALUATION AND ANALYSIS

IDEAL provides an experimental testbed to evaluate our theory of innovative design. In this chapter, we describe the evaluation of our theory at two levels: the theory as a whole and the individual “components” of the theory.

At the level of the theory as a whole, our theory of innovative design concerns with three core issues: (1) non-local modifications to known designs, (2) cross-domain transfer of design knowledge, and (3) reformulation of design problems. The evaluation question at this level is how well our theory accounts for these three issues. That is, (1) what classes of non-local modifications can it cover and what it cannot, (2) what classes of cross-domain transfer can it account for and what it cannot, and (3) what classes of problem reformulations can it cover and what it cannot? We have evaluated our theory along the following dimensions:

- Computational feasibility
- Generality in the sense of coverage of different tasks and domains, in particular, the following 4 types of coverage:
 - Coverage of different types of design adaptation
 - Coverage of different tasks in analogy
 - Coverage of different types of analogies
 - Coverage of different domains
- Common representations in two different ways:
 - For different types of models
 - Across different processes in analogical design
- Scalability

For the purpose of evaluating the theory at the level of individual components, our theory can be considered to have four main components: problem solving, learning of design patterns, qualitative modeling and learning of device models, and memory (i.e., its organization, indexing, and index learning). We have evaluated each of these components along different dimensions. Table 9.1 shows the dimensions of evaluation for each of these four components of our theory. We have used five different methodologies of evaluation listed below:

Table 9.1: Individual Components of Our Theory and The Dimensions of Evaluation for Each

Component of Our Theory	Dimensions of Evaluation
Problem Solving	Coverage of different design tasks Comparison of different adaptation strategies for efficiency Quality of solutions Use of the problem solver for a completely different task
Learning of Design Patterns	Coverage of different types of design patterns Expanded coverage of design problems Different interaction conditions in knowledge acquisition from external feedback Interactions of learning with other components of the theory
Qualitative Modeling and Learning of Device Models	Coverage of different types of models Different methods of acquiring device models Use of the same representations for a completely different task
Memory: Organization, Indexing, and Index Learning	Efficiency (of retrieval) Use of the indexing scheme for retrieval in a completely different task

- **Testing the computational feasibility**—where we not only make sure that different parts of the theory are implemented and operational in a computer program, but also include the testing involved in establishing the coverage of different classes of problems, tasks, and domains.
- **Mapping of design decisions to program (system) behavior**—that is, various kinds of ablation studies where we also include the comparative evaluations between different strategies and different types of knowledge.
- **Changing the interaction conditions**—with the external world as well as among the different components of the theory.
- **Changing the input conditions**—for different components of the theory, especially learning.
- **Trying the program on a completely different task** (than it is originally designed for) such as Knowledge Acquisition from Natural Language—in other words, subjecting the theory to new constraints arising from new task contexts.

9.1 Evaluation of the Theory as a Whole

In this section, we will describe how our theory is evaluated along specific dimensions that are applicable to the theory as a whole. We will defer our discussion of how well our model-based theory of innovative design accounts for the three facets of innovative design until the end of the chapter.

9.1.1 Computational Feasibility

Our model-based theory of innovative device design is computationally feasible as it is implemented in IDEAL. IDEAL has been tested with 50 different designs from 5 different domains for different aspects of the theory: 28 designs were used to test the learning of 6 different GTMs and the use of 3 of them; 4 designs were used to test the learning and use of GPPs (at different levels of abstraction); and 20 designs were used to test the model-based indexing, index learning, and retrieval. These designs were also used to test the other components of the theory in IDEAL along the specified dimensions of evaluation.

9.1.2 Generality

By generality of a theory, we mean how well it covers different types of tasks, different types of knowledge, and different domains. In particular, we describe below how our theory is general along four different dimensions:

1. Coverage of Different Types of Design Adaptation

2. Coverage of Different Tasks in Analogy
3. Coverage of Different Types of Analogies
4. Coverage of Different Domains

9.1.2.1 Coverage of Different Types of Design Adaptation

IDEAL's theory covers not only adaptive design tasks but also redesign based on external feedback from an evaluation of designs. IDEAL demonstrates the use of GTMs in adaptive design task and the use of GPPs in redesign based on evaluation.

9.1.2.2 Coverage of Different Tasks in Analogy

This means generality of the theory in that it covers multiple tasks in analogical design. For instance, IDEAL addresses both learning and use of design patterns in analogical design; it also addresses the tasks of retrieval of source analogues, evaluation of new designs, and learning of indices for design analogues and design patterns for storing them in memory. Thus an important aspect of IDEAL's evaluation is that it addresses several interesting and hard issues in analogical design. It is important to consider different issues because solutions to each imposes constraints on the others; otherwise, the resulting theories may be underconstrained. For example, the memory processes impose constraints on the kinds of problem solving that can be supported, problem-solving processes impose constraints on the kinds of learning that are needed, and the learning processes impose constraints on what is available in memory.

More specifically, the indexing scheme for design analogues in memory strongly constrains the class of design problems that can be solved and also constrains the retrieval processes. For instance, indexing only by functions of designs limits the class of problems to one where functional specifications are given. Since the next step is the transfer of source design analogue to the target problem, the types of source designs that can be retrieved puts constraints on the processes of transfer and modification by specifying the types of modifications that are needed for successful problem solving. Similarly, the processes of transfer also put constraints back on the retrieval. In addition, covering both retrieval and transfer brings out the issue of interaction (and control) between the two stages which will not surface if only either stage alone were modeled; and thus it constrains our theory more. Similarly, covering the stage of evaluation of the target analogue puts different requirements on the previous stage of transfer and vice versa. IDEAL also covers the issue of learning abstractions from the source and target design analogues, and as a result, it imposes constraints on the stages of transfer and modification, and evaluation in terms of the information that those stages have to generate as input to the learning stage. Finally, if the storage of target design analogue and learning of indices are also covered, they bring out the interactions with both indexing of design analogues in memory and

their organization. For instance, index learning can relax certain assumptions about memory, such as not requiring the complete knowledge of indices to source analogues a priori.

9.1.2.3 Coverage of Different Types of Analogies

Within a given characterization of what a “domain” is, IDEAL covers different types of analogies ranging from within-domain to cross-domain. For instance, it can use its design patterns learned in one domain (say, electric circuits) to solve adaptive design problems in the same domain and to solve problems in a different domain (say, heat exchangers) as well. We have varied the domains in which IDEAL learns the design patterns and tested whether it could successfully use the learned patterns in the others (including the one in which it learns). As we describe in the next section, IDEAL could learn the design patterns in different domains and use them successfully in different, other domains as well as it could in the same domains.

9.1.2.4 Coverage of Different Domains

This is to ensure that the theory is applicable in multiple domains. IDEAL has been tested in five different domains, namely, the domains of simple electric circuits, heat exchangers, electronic circuits, mechanical devices (including momentum controllers and velocity controllers), and coffee makers for both learning and use of design patterns. IDEAL can learn the design patterns in different domains and also use them in different domains. For instance, it can learn the feedback mechanism in the domain of electronic circuits and use it in the domain of mechanical controllers, and vice versa. Similarly, IDEAL could equally use the cascading mechanism in the domains of simple electric circuits, heat exchangers, and electronic circuits (with op-amps), irrespective of the source domains (i.e., simple electric circuits including devices of flashlight circuits and electric heaters, and heat exchangers) in which it had learned the mechanism.

9.1.3 Common Representations

This is to establish the adequacy of the same representations, for instance, SBF models, in supporting a number of different processes and tasks such as the generation of a design, evaluation of a design, and learning of different types of design patterns.

9.1.3.1 For Different Types of Models

IDEAL uses the same SBF language to represent both models of devices and models of design patterns. The representation of design patterns is a graceful extension of SBF models.

9.1.3.2 Across Different Processes in Analogical Design

An important aspect of IDEAL’s evaluation is the uniformity of its representations across different stages of analogical design. IDEAL uses the same representations of device models for

supporting different stages. Of course the inferences drawn, and, therefore, the functional role played by the models changes from one stage of processing to another. The functional part of SBF models acts as index into the memory of analogues, and this enables IDEAL to infer similarity between the target problem and the stored analogues in the retrieval stage. The SBF model of a source design provides the functional and causal explanations of how the design works and this enables IDEAL to infer the parts of the design that need to be repaired in the transfer and modification stage. SBF models enable design verification by qualitative simulation in the evaluation stage. The functional and causal explanations that an SBF model provides of how the target design works enables IDEAL to learn abstractions in the next stage of analogical design, and so on.

9.1.4 Scalability

IDEAL now contains 50 design analogues from 5 different domains. It learns eight different design patterns that fall into two different types. The largest design in IDEAL has the orders of 10 structural elements and 10 structural relations, and the order of 3 inter-dependent behaviors in its SBF model. That is the design of a gyroscope control system with feedback. Although IDEAL is thus not-a-small system, the scalability of the model-based approach still remains an open issue.

9.2 Evaluation of the Individual Components of the Theory

In this section, we will describe how the individual components of our theory are evaluated along different dimensions (Table 9.1) that are appropriate for each.

9.2.1 Problem Solving

In the problem-solving component of our theory, we include from our computational process of model-based analogy (Figure 2.1) the stages of analogical transfer and modification, and evaluation of the solution. We describe our evaluation of this component of our theory along four different dimensions:

1. Coverage of Different Design Tasks
2. Comparison of Different Adaptation Strategies for Efficiency
3. Quality of Solutions
4. Use of the Problem Solver for a Completely Different Task

An analysis of our methods for the tasks in problem solving indicates that they are efficient. For instance, our method for the spawning of adaptation goals (i.e., the diagnosis of the source

design analogue) is efficient because the organization of the SBF model of the source design enables quick localization of the search for “faults” in the given design to a small portion of the SBF model, yet enabling non-local modifications due to the instantiation of appropriate GTM.

9.2.1.1 Coverage of Different Design Tasks

Under this dimension, we are interested in finding and characterizing the classes of design problems that our theory of adaptive design covers. In our theory, the ability to make modifications to known designs is determined by the knowledge of the adaptation strategies (whether those strategies are learned or given). Some of the adaptation strategies make use of the knowledge of design patterns (in particular, GTMs) and some others do not. While describing how our theory covers different non-local modifications, we will later give in Table 9.2 a characterization of the types of non-local modifications that each of the GTMs enables. In addition, we will also indicate what type of non-local modifications that the component-addition strategy can enable. That, we hope, will indicate to some extent the classes of adaptive design problems covered in our theory. We will now try to characterize for each GTM the class of problems it enables in a more formal specification. That is, we specify the class of problems in terms of templates for the functions of candidate designs and desired designs, and the conditions under which the particular GTM is applicable. Figures 9.1, 9.2, and 9.3 respectively show such characterization of the classes of problems for component-replacement strategy, GTM of one type of feedback, and GTM of another type of feedback.

In addition to being able to make non-local modifications, IDEAL can also do some simple modifications to known designs. Although the class of problems that the simple adaptation strategies enable are not as interesting, they nevertheless contribute to the coverage of design problems. Just to reiterate from Chapter 5, IDEAL has these four simple adaptation strategies: Substance Substitution, Substance-Property Modification, Component Replacement, and Component-Parameter Modification. Each of these strategies enables IDEAL to solve a different class of adaptive design problems.

9.2.1.2 Comparison of Different Adaptation Strategies for Efficiency

When there are multiple, alternative types of knowledge that can be used to solve the same class of problems, problem-solving efficiency is one criterion that may be used to compare those alternatives for their merits and demerits. Of the different GTMs this research has explored, only the cascading mechanism has an alternative strategy that can be used to solve the same class of problems. That is the strategy of component addition. Hence this discussion centers around only the cascading mechanism and component addition. We measure the problem-solving efficiency in terms of the number of steps of modification (and evaluation) required to successfully generate a solution. Each step involves both making the modification to the source design and verifying if that modification results in the desired function.

We have tested IDEAL for the comparison of these two adaptation strategies: instantiation of cascading mechanism and component addition. In this testing, we found that these two adaptation strategies have equal competence in solving the same class of design problems but different performance. The use of the cascading mechanism enables IDEAL to solve any problem from the same class in one step of modification (and evaluation), while the use of component addition requires n steps (where n is a measure of the size of the adaptive design problem). Therefore, by learning the cascading mechanism, IDEAL becomes more efficient at solving the same class of problems.

We have described in Chapter 8 how the component addition strategy can enable IDEAL to solve the design problems from which it can learn the cascading mechanism. In applying the component addition strategy, generating a design for a new desired function requires only one step when there exists a component whose function is same as the difference between the desired and that of one in the retrieved design analogue; but it will require more than one step to generate a design if the available components only achieve functions that are $<$ the difference. That is, for instance, when the desired function is to achieve 3 volts and an available component achieves 1.5 volts, component addition suggests that a component that achieves 1.5 volts can be added to the available one; similarly, as we illustrated in Chapter 8, when the desired function is to achieve 6 volts and an available component achieves 1.5 volts, it requires three steps, adding one 1.5-volt battery in each step. Thus, with the knowledge of component addition, IDEAL could solve the same class of problems as it could with the cascading mechanism when all the following conditions hold good: (1) there exists a design analogue in which the component selected for modification has a parameter value that is *smaller than* the parameter value desired, as determined by the new range of transformation in the desired function; (2) there are components whose parameter values are \leq the difference (new parameter value – the smaller parameter value); (3) such components can be selected in some combination so that the sum of their parameter values is equal to the difference (new parameter value – the smaller parameter value); and (4) the component parameter values are additive when connected in series and so are their ranges of functional transformation.

From the above description of the component addition strategy, it is clear that for all the problems to which the cascading mechanism applies the component addition also applies. However, generating a design for a problem in this class using the component addition strategy requires n iterations of its application and evaluation (i.e., n steps of modification and evaluation) where n is the number of replications of a component needed (that is, the desired function $F_2 = n * F_1 + F_3$ where F_1 is the function of a component being modified in the retrieved design analogue and F_3 is the function of a similar type component that specifies a smaller range of transformation than what F_1 does). Whereas, generating a design for every problem in the same class using the mechanism of cascading requires only *one* step of its application and evaluation, independent of n . Therefore, learning the cascading mechanism (and using it) significantly improves IDEAL's problem-solving efficiency for the same class of problems in comparison to using

the component addition strategy.

In the above comparison, we only considered the common class of problems to which both the strategies apply. But there are differences in the coverage of problems each strategy potentially provides! Since, in general, for any iterative approach (i.e., repeated application of a strategy) there needs to be a finite upper bound on the number of iterations one could do using the same knowledge, there could be design problems (for which $n >$ the finite upper bound) that may not be solvable using the component addition strategy alone. However, using the cascading mechanism for those problems could lead to successful designs. At the same time, the component addition strategy facilitates solving certain design problems that the cascading mechanism alone cannot. Those problems have a desired function F_2 in relation with the function of the available component such that $F_2 = F_1 + F_3$ where F_3 is smaller than F_1 ; in other words, there is no replication of the available component required at all in solving these problems! This class may also include problems where F_3 is smaller or larger than F_1 and the structural constraints specify that there is only one component available to achieve F_1 (i.e., the structural constraint inhibits replication of the component for F_1 even when $F_3 > F_1$).

9.2.1.3 Quality of Solutions

In adaptive design, one way the quality of a solution can be measured is in terms of the number of components used in the design for a given problem. In our theory, the quality of a solution generated depends on the available knowledge in device models and what inferences that knowledge can enable. For instance, a model described in detail can enable localization of a needed modification to a primitive component while a model that describes the behavior of a device only at a high level cannot. When a modification to a known design in order to achieve new functions can be localized to a primitive component, the resulting solution can be of better quality (i.e., it can be more parsimonious in terms of the number of components used); of course, it also depends on the particular adaptation strategy used. For example, using the component addition strategy may result in more parsimonious designs than using the instantiation of the cascading mechanism because the latter suggests the use of several components to achieve a function while the former suggests the addition of one component.

KRITIK2, the earlier version of IDEAL, has been specifically evaluated for the quality of solutions as follows (Stroulia and Goel, 1992): the question was how is the quality of solutions affected by the detail in the known device model when the adaptation strategy available is the instantiation of the cascading GTM. That evaluation led to the finding that the more detailed the known device model is, the more the quality of solutions generated by using the cascading GTM.

9.2.1.4 Use of the Problem Solver for A Completely Different Task

Initially, the theory of our problem-solving component has been developed for the task context of design. In this task context, the design problems are specified in SBF representations. But, later in a companion project called KA (Peterson et al., 1994), we have used the same problem solver¹ for solving problems specified in Natural Language and for understanding design descriptions in Natural Language. We found that the same theory was equally applicable in that task context as well.

9.2.2 Learning of Design Patterns

We evaluated this component of our theory along the following different dimensions:

1. Coverage of Different Types of Design Patterns
2. Expanded Coverage of Design Problems
3. Different Interaction Conditions in Knowledge Acquisition from External Feedback
4. Interactions of Learning with Other Components of the Theory

An analysis of our methods for learning indicates that they are efficient because the organization of the SBF models together with the problem-solving context provides constraints on learning of design patterns and the comparison of design analogues takes place only on small, focused behavior segments in their models.

9.2.2.1 Coverage of Different Types of Design Patterns

Learning and Use of GTMs: IDEAL covers six different GTMs: one cascading GTM, four types of feedback GTM, and one type of feedforward GTM. It could learn all these six different GTMs using the same model-based learning method—it could learn them by abstracting over the patterns of regularity in device models of the design analogues. Furthermore, IDEAL could recognize the relevance of the appropriate GTM (of the three on which it was tested), retrieve it, and instantiate it in the context of adaptive design scenarios. By instantiating the retrieved GTMs in the SBF models of the source designs, IDEAL produced solutions to the target problems.

Learning and Use of GPPs: IDEAL covers two different GPPs (the physical process of heat exchange at a lower level of abstraction and the physical process of heat flow at a higher level of abstraction—the difference between them being in the specification of the amount of specific structural information from the devices of cooler and heater from which it learns them). Using the same model-based approach as above, but using a different strategy/algorithm as suggested by the type of regularity in the given design analogues, IDEAL could learn both GPPs from two

¹actually KRITIK2 which is an earlier version of IDEAL

designs, one of acid cooler and the other of water heater. We have tested IDEAL then for its use of the learned physical processes in the task of redesigning a failed coffee maker. It could successfully recognize the relevance of these GPPs and use them for forming causal explanations of the observed undesired behavior in the coffee maker. It could subsequently redesign the coffee maker to compensate for the failures. Although both the physical processes (heat exchange and heat flow) were applicable in the context of the coffee maker redesign, using the process of heat exchange required more specific elements (such as heat-exchange chamber, pipes and the relationships among them) in the model to be adapted to the target design of coffee maker than those in using the process of heat flow. Thus, the higher the level of abstraction of a GPP, the lesser the adaptation needed in order to transfer it to a new design situation.

9.2.2.2 Expanded Coverage of Design Problems

The larger the coverage of tasks in the domain(s) of consideration the better accepted and more general is a theory. Since IDEAL's design task is really adaptive design, the design tasks can be characterized by the differences in the functions of the known, retrieved design and the desired design that need to be reduced. Under the knowledge conditions that IDEAL knows only simple adaptation strategies such as component replacement and substance substitution, it can only solve the class of problems that involve simple, parametric differences. However, its acquisition of each new GTM enables it to solve a different class of problems, where each class is characterized by a different type of functional difference. That is, learning of GTMs increases the coverage of problems in IDEAL.

Consider for instance the cascading mechanism. Before learning it (or in general without the knowledge of the cascading mechanism), under the above knowledge conditions, IDEAL could only solve a simple class of design problems. A design problem in this class has a functional specification of the form shown in Figure 9.1, which is of type *substance-property transformation*. The state of analogue memory is such that it has a design analogue whose range of functional transformation ($V_1 \sim V_2$, where V_1 , V_2 are respectively the input and output values of the substance in the function) is dependent on some parameter of a component in the design.² Without the knowledge of the cascading mechanism, IDEAL could solve this class of design problems *only if* the range of transformation desired is same as the range of transformation in a matching design analogue or there exists a component of the desired parameter value as determined by the new range of transformation. For instance, a device to achieve 0-3 volt transformation function can be designed only if there exists a battery with a 3-volt capacity.

After learning the cascading mechanism, however, IDEAL could solve a larger class of design problems (including the above shown in Figure 9.1) whose specifications may also involve structural constraints of the kind that specific components with specific parameter values are not available (or are non-standard). In general, these structural constraints can be part of the

²In general, the memory can have a class of design analogues each of which satisfies this condition.

DESIRED DESIGN:**GIVEN:** F_2 **MAKES:**

?SUB ?prop1: ?val11

?SUB ?prop1: ?val22

There exists a

CANDIDATE DESIGN:**GIVEN:** F_1 **MAKES:**

?SUB ?prop1: ?val11

?SUB ?prop1: ?val21

such that

CONDITION: $?val22 \neq ?val21$ $(?val11 \sim ?val21) \mathbf{R}$ some component parameter

Figure 9.1: The Class of Problems IDEAL Can Solve BEFORE It Learns the Cascading Mechanism

problem specification, or they can also come from the memory of available components. That is, for instance, a design problem may specify that the new design to achieve 3 volts cannot use a 3-volt battery. Alternatively, the memory of available components may not have a 3-volt battery because it is non-standard. With the knowledge of the cascading mechanism, IDEAL could solve this new, larger class of design problems under certain conditions: (1) there exists a design analogue in which the component selected for modification has a parameter value that is *smaller than* the parameter value desired as determined by the new range of transformation in the desired function; (2) there are sufficient number ($= \lfloor (\text{new parameter value} / \text{smaller parameter value}) \rfloor$) of components of the smaller parameter value; (3) when the residue of the ratio ($\text{new parameter value} / \text{smaller parameter value}$) is non-zero, there exists a component of the parameter value equal to the residue; and (4) the component parameter values are additive when the components are connected in series and so are their ranges of functional transformation.

Similarly, we can specify for each of the other GTMs, namely, feedback, feedforward, and device composition, the class of problems “before” and “after” learning. It is clear for each of these generic mechanisms that the class of problems IDEAL could solve after learning that generic mechanism is larger than those that it could solve before. For instance, Figure 9.2 illustrates the class of problems that the feedback mechanism of one type enables. In this type of feedback mechanism, the input, output, and feedback substances are all the same. Before learning this type of feedback mechanism, IDEAL cannot solve the illustrated class of problems. Similarly, Figure 9.3 illustrates the class of problems for a second type of feedback; in this type, although the input substance and the feedback substance are the same, the output substance is different from the feedback substance. Again, IDEAL cannot solve the illustrated class of problems before learning the second type of feedback mechanism.

9.2.2.3 Different Interaction Conditions in Knowledge Acquisition from External Feedback

IDEAL has been tested for its learning of design patterns under different interaction conditions with an oracle that provides external feedback upon a problem-solving failure. When IDEAL fails to solve design problems, it is given external feedback from which it learns the GTMs. We have varied the information presented as feedback and observed in those different learning situations what processes might be involved in IDEAL’s learning and whether it can learn at all. In all the four different interaction conditions with which IDEAL was tested, we found that it could learn the GTMs. However, it required to make more inferences as the information in the external feedback reduced. Three interaction conditions involved presentation of

1. desired design for the target problem and an SBF model for the solution or
2. only the desired design for the problem or
3. only the solution to the local, specific adaptation goal.

DESIRED DESIGN:

GIVEN:	?SUB ?prop1: ?val11
F₂	
MAKES:	?SUB ?prop1: ?val22

There exists a

CANDIDATE DESIGN:

GIVEN:	?SUB ?prop1: ?val11
F₁	
MAKES:	?SUB ?prop1: ?val21

such that

CONDITION:

$$\begin{aligned} ?val22 \neq ?val21 & ; ?val21 = ?val \pm \Delta \\ ?val22 &= ?val \pm \delta \end{aligned}$$

The fluctuations in the output substance-property value R a single component parameter.

$$\begin{aligned} F_2 &= f : (?val11, ?val21) \rightarrow ?val11' \\ &+ F_1 : (?val11') \rightarrow ?val21 \end{aligned}$$

and there exists a design for achieving f , and $?val11'$ is the substance-property value at an intermediate state in the candidate design behavior.

$$\begin{aligned} ?val11' &= f_+(?val11) \\ ?val11' &= f_-(?val21) \end{aligned}$$

Figure 9.2: The Class of Problems IDEAL Can Solve AFTER It Learns One Type of Feedback Mechanism

DESIRED DESIGN:

GIVEN:	?SUB
F₂	?prop1: ?val11
MAKES:	?SUB
	?prop1: ?val22

There exists a

CANDIDATE DESIGN:

GIVEN:	?SUB
F₁	?prop1: ?val11
MAKES:	?SUB
	?prop1: ?val21

such that

CONDITION:

$$\begin{aligned} ?val22 \neq ?val21 & ; ?val21 = ?val \pm \Delta \\ & ?val22 = ?val \pm \delta \end{aligned}$$

The fluctuations in the output substance-property value **R** a single component parameter.

$$\begin{aligned} F_2 &= f : (?val11, ?val11'') \rightarrow ?val11' \\ &+ F_1 : (?val11') \rightarrow ?val21 \\ &+ g : (?val21) \rightarrow (?val22, ?val11'') \end{aligned}$$

and there exist designs for achieving **f** and **g**, and **?val11'** is the substance-property value at an intermediate state in the candidate design behavior. Similarly, **?val11''** is the value at a new state.

$$\begin{aligned} ?val11' &= f+(?val11) \\ ?val11' &= f-(?val11'') \\ ?val11'' &= f+(?val21) \end{aligned}$$

Figure 9.3: The Class of Problems IDEAL Can Solve AFTER It Learns A Second Type of Feedback Mechanism

The fourth condition, in contrast, involved no interaction with the oracle; instead, IDEAL is given alternative adaptation strategies such as component addition that enabled it to solve the problems but it took more modification steps than necessary for some problems. In the following sections we use the cascading mechanism as an example GTM to describe the evaluation under these conditions.

Interaction Condition 1: When Oracle presents the Desired Design and its SBF Model This is the simplest condition in terms of the inferences needed on IDEAL's part in order to learn the GTMs. Since the SBF model for the desired design is also given by the oracle, IDEAL only needs to compare the models of the source design (that it was modifying) and the desired design. We have described this process in sufficient detail in Chapter 7. IDEAL could learn all the cascading, feedback, and feedforward mechanisms under this condition.

Interaction Condition 2: When Oracle presents only the Desired Design Since IDEAL requires the SBF models of the source and target design analogues in order to form any abstractions, under this condition, it requires an additional inference step that involves deriving the behavior for the given structure of the desired design. Hence, given the structure and the function (which is already available as part of the problem specification) of the target/desired design, IDEAL first derives the internal causal behaviors of the given structure in order to comprehend how the given structure achieves the desired function. Once it comprehends the functioning of the given structure in terms of internal causal behaviors of the structure, it compares the new behavior with the behavior of the source design analogue and forms any generic mechanisms it can discover. We have tested IDEAL for its learning of the cascading and feedback mechanisms under these knowledge conditions in the domains of electric circuits, heat exchangers, and electronic circuits (with operational amplifiers). We have described IDEAL's method and illustrated its generation of an SBF model in the context of learning the cascading mechanism in Chapter 8.

Interaction Condition 3: When Oracle presents only the solution to the local, specific adaptation goal In this interaction condition, the information presented to IDEAL by the oracle upon problem-solving failure is much less compared to the information in Conditions 1 & 2. As a result, IDEAL has an additional inferential burden in learning generic mechanisms. The input knowledge conditions in this situation are such that when IDEAL fails to generate a design for a new problem it is given only the localized structure for a correct target design. When IDEAL fails, it will also have the knowledge of the old, localized structure that it could not modify. However, in order to learn a generic mechanism in this situation, IDEAL needs to make two additional inferences. The first is to revise the structure of the source design to include the given new localized structure, and the second is to revise the behavior of the source design to generate the internal causal behaviors for the new structure. Once IDEAL generates the structure for the target design problem, it can generate its internal causal behaviors and then compare the behaviors of the source design analogue and the target design to form any generic mechanisms it can discover. We have tested IDEAL for its learning of the cascading

and feedback mechanisms under these knowledge conditions in the domains of electric circuits, heat exchangers, and electronic circuits (with operational amplifiers). We have described and illustrated IDEAL's process for the generation of structure of a new design from a given localized structure in the context of learning the cascading mechanism in Chapter 8.

Interaction Condition 4: When there are no problem-solving failures This condition is quite distinct from the other three on which we have tested IDEAL because it involves *no* interaction with the oracle and *no* problem-solving failure. That is, IDEAL need not receive any information from the oracle in order to learn some generic mechanisms under certain conditions. Also, it need not fail and receive feedback for learning. Instead, it can also learn from successful, but inefficient, problem solving. Obviously, IDEAL needs to be given some alternative strategies for adaptation using which it can solve the given design problems. However, using this alternative knowledge, it may take more modification steps than necessary. This implies that the alternative strategies provide IDEAL with the same *competence* as would any "new" generic mechanisms that may be learned. But the new generic mechanisms learned in this situation may make a difference in the *performance* of IDEAL, that is, they enable IDEAL to solve similar design problems faster. Thus IDEAL's learning task in this situation is equivalent to knowledge compilation.

To the extent this research has explored, there were no alternative strategies for feedback, feedforward, and device composition. But for the cascading mechanism, an alternative is component addition. Therefore, we have tested IDEAL only on learning of the cascading mechanism under this interaction condition. In Chapter 8, we have described and illustrated how IDEAL generates the new design and its SBF model by taking a number of inference steps where the number is proportional to the size of the problem with respect to the source analogue (i.e., n is the ratio of the desired component parameter and the available component parameter where the parameters are determined by the difference in the functions of the candidate and desired designs).

9.2.2.4 Interactions of Learning with Other Components of the Theory

In this section, we describe our evaluations of IDEAL's learning of design patterns (in particular, the learning of the cascading GTM) for the interactions with other subtasks of analogical design and knowledge conditions. We have evaluated along six different dimensions:

1. Influence of Design Analogue Selection on Learning
2. Influence of the Order of Presentation of Design Situations on Learning
3. Learning from Different Sets of Specific Design Situations
4. Learning from Design Examples with Different Structural Configurations
5. Influence of the Representation of SBF models on Learning
6. Influence of the Internal Organization of SBF models on Learning

All the following descriptions are in the context of examples presented in Chapter 7, i.e., learning of the cascading mechanism from the designs of 1.5-volt electric circuit (EC1.5) and 3-volt electric circuit (EC3), and learning of the feedback mechanism from the designs of a simple amplifier and an inverting amplifier.

9.2.2.4.1 Influence of Design Analogue Selection on Learning Recall from Chapter 7 (Section 7.2.4) that when the problem of EC4 was presented,³ both designs EC1.5 (Figure 7.2) and EC3 (Figure 7.3) were retrieved because their functions were all similar. In that section, we supposed that EC1.5 was selected and described how the initially hypothesized cascading mechanism (Figure 7.6: representation in (a) and the shaded region of (b)) would be revised based on EC1.5 and EC4. It is however interesting to explore what the processing would have been and what the outcome would have been if EC3 were selected. Using the functional difference between the problems of EC3 and EC4 and diagnosing the SBF model of EC3 (as per the process described in Chapter 5) results in either of the batteries in EC3 as possible candidates for modification. Selecting either of the batteries for modification is equivalent. Hence let us suppose that *Battery1* is selected. The functionality of *Battery1* can be informally described as producing electricity with a voltage of 1.5 volts. The new desired voltage is 2.5 (i.e., $4 - 1.5$) volts because *Battery2* already provides 1.5 volts. Both the component-replacement and the learned cascading mechanism are applicable in this situation to modify the retrieved design; but the latter is more applicable than the former because of the structural constraints specified. The difference between the functions of *Battery1* and the new desired component match the functional-difference index of the cascading mechanism, but the decomposability condition on the desired function cannot be satisfied (i.e., $2.5 \neq n * 1.5$ for any integer n). Therefore IDEAL fails to generate a design for the new problem.

Suppose now that a correct solution for the new problem (Figure 7.7) is given. Then, IDEAL's learning is triggered. The above problem-solving context enables IDEAL to focus on the behavior segments $state_{1-2} \rightarrow state_{1-3}$ which is B_1 (in Figure 7.3(b)) and $state_{1-2} \rightarrow state_{1-4}$ which is B_2 (in Figure 7.7). By comparing these two segments according to the model-based learning method described in Chapter 7, IDEAL finds that B_1 matches with the segment $state_{1-2} \rightarrow state_{1-3}$ in B_2 and that there is a succeeding segment in B_2 that does not match with B_1 . Note that n , the number of segments in B_2 that match with B_1 , is now 1. As it can be seen, the regularity situation is as in 1 in Table 7.1. Therefore, IDEAL learns a different generic mechanism as shown in Figure 9.4, which is a device-composition mechanism. This will not unify with the existing cascading mechanism, as it should not be, according to our theory because their indices won't unify. (See Section 7.2.4 for a description of when two GTMs unify.) That is, the decomposability conditions of F_2 in the two mechanisms won't unify because they specify qualitatively different values for n ($n = 1$ vs. $n > 1$). It is interesting to note that although both

³That is the problem of designing a circuit that takes 4 volts of electricity as input and produces 16 lumens of light.

EC3 and EC4 have instances of the cascading mechanism, the current problem-solving context and the focus have led to learning a different generic mechanism. Thus the selection of a design analogue can potentially influence what GTMs may be learned.

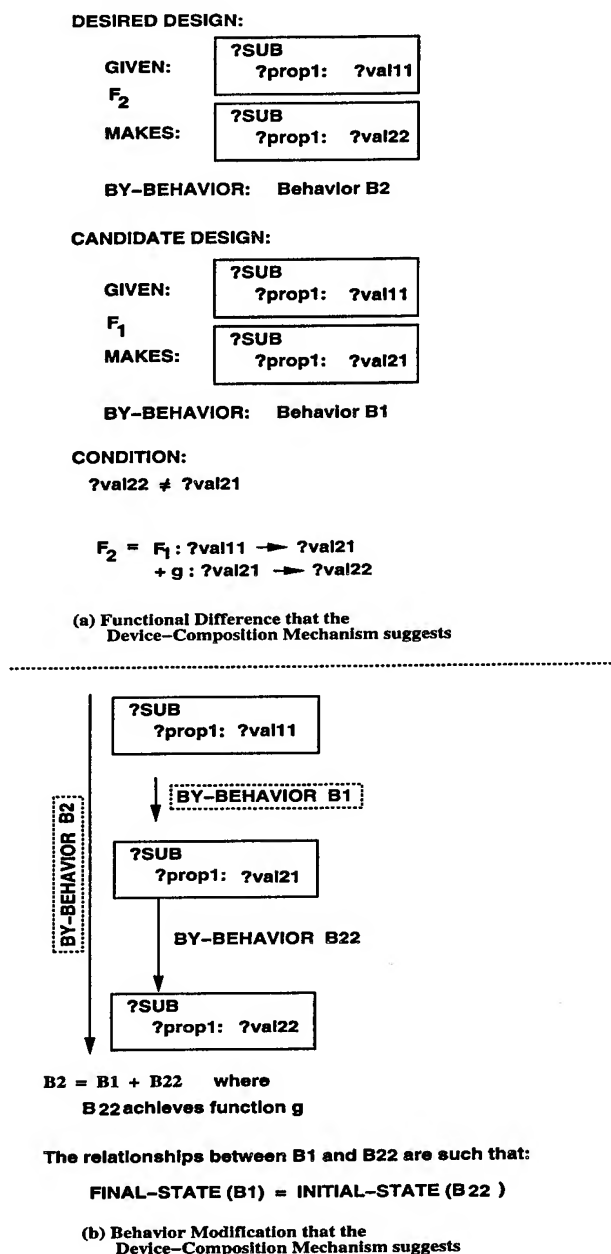


Figure 9.4: An SBF Representation of the Device-Composition Mechanism

Suppose that the focus was on the entire substructure consisting of both batteries in EC3 and all three batteries in EC4. In such conditions also, IDEAL's learning process results in

learning the device-composition mechanism as opposed to learning the cascading mechanism! This is because the comparison of the two focused behaviors was at the level of the entire focused behavior of the existing design (battery substructure in EC3) against the entire focused behavior of the new design (battery substructure in EC4). In contrast, if an external bias were given to compare at the level of a unitary behavior segment (i.e., a state-transition sequence consisting of two states and a single transition between them), then IDEAL could learn the cascading mechanism (as shown in Figure 7.6)! Then one might wonder why such a bias is not incorporated in our design of the model-based learning method! There are at least two reasons why it is not part of the method: (1) In general, for any two given design analogues, comparing at the level of a unitary behavior segment is computationally expensive and it results in a number of pairs of matches that can be combinatorial—the learner can be overwhelmed with the number of regularities! and (2) Even if the learner were to go through all possible regularities, it will postulate either the same set of generic mechanisms again and again, or such generic mechanisms that may never be useful in later design problem solving!

9.2.2.4.2 Influence of the Order of Presentation of Design Situations on Learning

Let us consider the effects of presentation ordering in just two design situations, i.e., the problems of EC3 and EC4. Suppose now that the design problem of EC4 is presented before that of EC3. Given the knowledge conditions as they were at the time EC3 was presented (Section 7.2.1) in the previous order of presentation, and given the problem of EC4, IDEAL would retrieve the design of EC1.5. Furthermore, it fails to generate a design due to the knowledge conditions and the structural constraints specified in the design problem. Given the correct solution for the problem of EC4 like earlier, IDEAL would hypothesize by following the process of model-based learning the model of cascading mechanism shown in Figure 7.6 (including both the shaded and unshaded representations in the figure). Now, presenting the problem of EC3 does not change the learned model of the cascading mechanism because (1) the learned cascading mechanism becomes applicable and its application results in a correct design for the problem of EC3 and (2) the cascading mechanism that can be hypothesized from EC1.5 and EC3 is anyway subsumed by the model learned from EC1.5 and EC4. Note that one design situation would be enough to learn the same description of the cascading mechanism if the right design situation were presented first, while it would require two situations in a different order! These are only specific situations, but the general point is that the model-based learning method provides the necessary competence which can result in the highest performance under the right situation(s). Thus the order of presentation of design situations can affect the speed of learning GTMs.

9.2.2.4.3 Learning from Different Sets of Specific Design Situations

Of course, the training examples do determine what exactly can a learner learn. But, the question is how dependent is the learner on the *specifics* of the examples. Within each source domain in which IDEAL learns the generic mechanisms, we have tested it for learning from different sets of specific

design situations. Consider, for instance, the design situations for the learning of the cascading mechanism. The different sets of design examples, we have tried, varied in the specific *component* that was repeated in the design, that is, battery in one set, resistor in another, and so on. We have tested IDEAL with the minimal number of training examples in each set. At the minimum, for each learning session, one design problem is presented and IDEAL learned from the target design and a source design analogue. The two design examples in each set are such that one does not have an instance of the mechanism to be learned and the other has an instance of the mechanism (e.g., a design of 1.5-volt electric circuit with one 1.5 volt battery and a design of 3-volt electric circuit with two 1.5 volt batteries connected in series).⁴

We have tested IDEAL with 10 different sets: four in electric flashlight circuits, one in electric flashlight circuits with explicit resistors, two in electric heaters with resistors, and three in heat exchangers. In all these situations except when the electric flashlight circuits with explicit resistors were presented did IDEAL learn the same partial model of the cascading mechanism! When the designs of electric flashlight circuits with explicit resistors were presented, it could not learn the cascading mechanism because the functions of the resistors in the two designs (i.e., the ranges of voltage transformation each resistor causes) were different. That is, in order for IDEAL to learn the cascading mechanism, the function of the particular component (in terms of a substance property value transformation) in one design should repeat more than once in the other with the same range of value transformation in each of the repetitions.

Thus, IDEAL's learning is independent of the specific components that repeat in the design examples but it is dependent only on the regularity in the SBF models of the designs. If the given designs do not have any of the regularities, obviously enough, it does not learn any generic mechanisms.

9.2.2.4.4 Learning from Design Examples with Different Structural Configurations

Design examples with certain regularities such as those for feedback and feedforward cannot have much flexibility in changing the structural configuration, but those with the replication of a component can have equivalent alternative structural configurations. Hence, we have tested IDEAL for how independent its learning of the cascading mechanism can be of the specific structural configuration in the design examples. For instance, we have tested if IDEAL could learn the same cascading mechanism from EC1.5 (Figure 7.2) and the design of EC3 with an alternative structural configuration (as shown in Figure 9.5) as it would learn from the design of EC1.5 and EC3 (Figure 7.3). Note that the two batteries in the alternative structural configuration of EC3 are not "contiguous," that is, they are not connected right next to one another. IDEAL indeed can learn the same cascading mechanism from two designs irrespective of the structural contiguity of the repeated components in the designs.

⁴That is, they have the structural regularity that can enable learning the cascading mechanism.

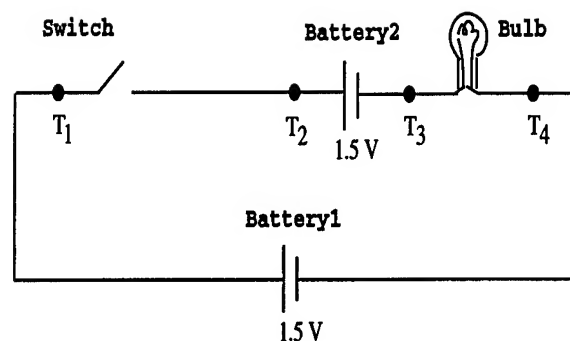


Figure 9.5: An Alternative Structural Configuration for the Design of EC3

9.2.2.4.5 Influence of the Representation of SBF models on Learning We have explored how important the representations of SBF models of the design examples are to learning of the generic mechanisms. In particular, we have experimented with the representation of states in the SBF models: states represented at the locations of structural connections *vs.* states represented at the level of components. All the examples so far presented have the representations of the former kind. The design analogue of EC1.5 is shown in the latter form of representation in Figure 9.6. Note that there are no states represented at the connections between Battery and Bulb, and between Battery and Switch; instead, a state is represented at the Battery specifying that electricity has a voltage of 1.5 volts.

As we have explained and illustrated before, with the SBF models represented in the first form, IDEAL could learn the appropriate generic mechanisms from the given design examples. With the second form of representations of SBF models, however, it could not learn the generic mechanisms. It is because in the second form of representing SBF models, the functions of the particular components that participate in the regularity are not explicit as a transformation of substance-property value. For instance, the representation of the SBF model of EC1.5 in Figure 9.6 does not indicate that the function of Battery is a transformation of voltage from 0 volts to 1.5 volts. This experiment establishes that in order for IDEAL to be able to learn, an important aspect of the representation of SBF models is that the functions achieved by each component in the design (that participates in the instance of the mechanism to be learned) need to be explicit in terms of a property value transformation.

9.2.2.4.6 Influence of the Internal Organization of SBF models on Learning We have tested IDEAL for how the internal organization of SBF models of the design examples affects learning of generic mechanisms. That is, in particular, we have experimented with two different internal organizations of SBF models: organization of behaviors of all the components in the device at the same, single level of detail (*i.e.*, a flat organization) *vs.* organization of behaviors in a hierarchy with behaviors of substructures at a lower level of detail (*i.e.*, in a

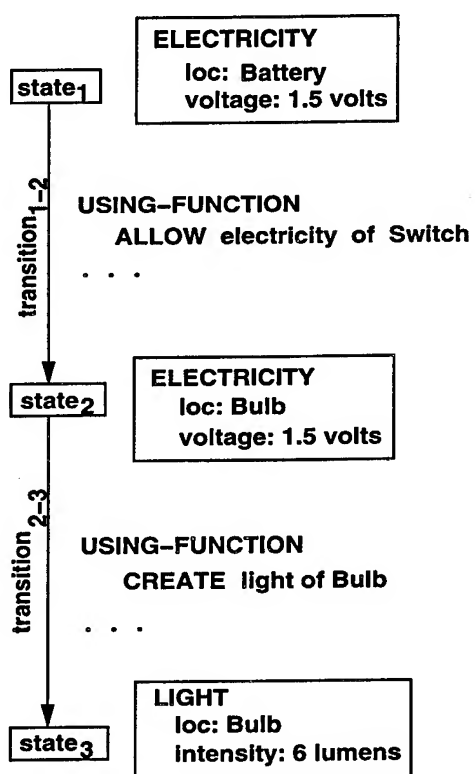


Figure 9.6: An Alternative Specification of the Behavior of EC1.5

two-level organization, behaviors of the substructures of a device are at the lower level and the overall behavior of the device is at the higher level).

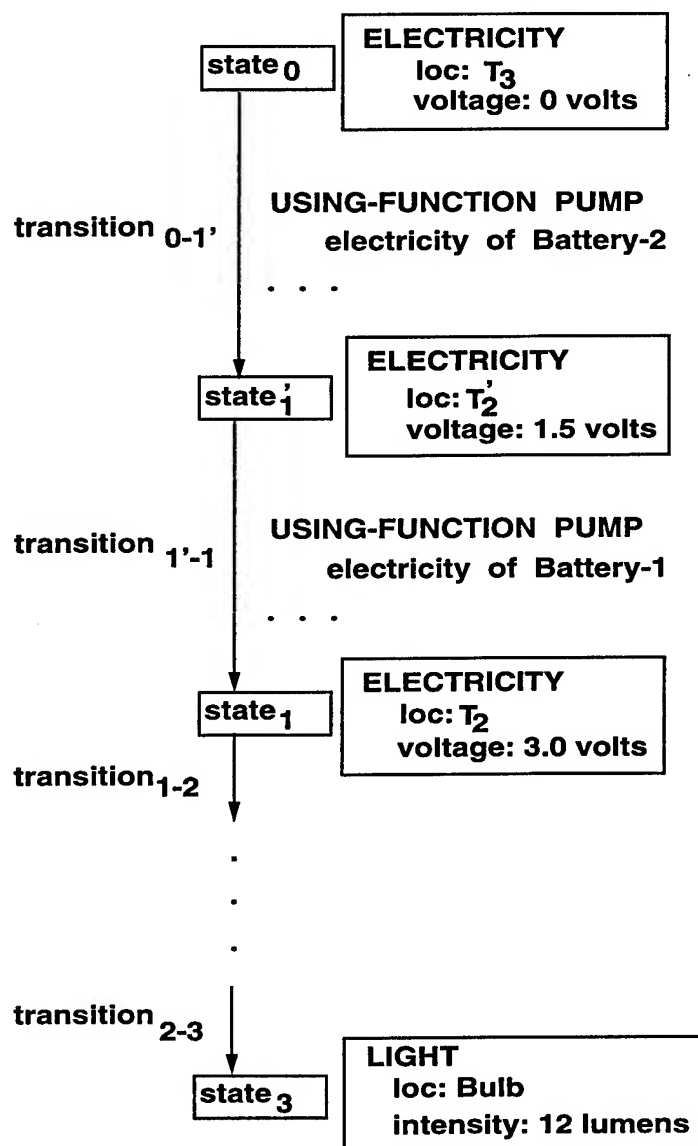


Figure 9.7: Flatly Organized Behavior of the Design of A 3-volt Electric Circuit

We have done this experiment with the design examples for learning of the cascading mechanism. The internal organization of the SBF models in EC1.5 and EC3 presented earlier are, for instance, in the second category. A flat organization of the behavior in the SBF model of EC3 is shown in Figure 9.7. We found that IDEAL can learn the cascading mechanism in the same way in both cases of the internal organization of SBF models. One would expect that

it may take much longer to find regularities in a flatly organized complex behavior of an SBF model, compared to that in a hierarchically organized behavior. However, since IDEAL exploits not only the internal organization of SBF models but also the problem-solving context to focus on the parts of the behaviors for abstraction, it only takes a little longer to focus in on the sub-behaviors in a flatly organized model than it would in a hierarchically organized model!

9.2.3 Qualitative Modeling and Learning of Device Models

This component of our theory has been evaluated in the following three dimensions:

1. Coverage of Different Types of Models
2. Different Methods of Acquiring Device Models
3. Use of the Same Representations for a Completely Different Task

9.2.3.1 Coverage of Different Types of Models

Our theory of qualitative models accounts for the content and representation of two different types of models: models of devices and models of design patterns. As mentioned earlier too, the SBF language we adapted from (Goel, 1989) is adequate to represent both these types of models.

9.2.3.2 Different Methods of Acquiring Device Models

Our theory accounts for multiple methods of acquiring device models:

1. by revision of known, similar models (Goel, 1991b)
2. by instantiation of design patterns in known models (while solving problems as explained in Chapter 5)
3. by a combination of model revision and primitive behavior composition (when structural specifications are given in the interaction with an oracle)

Earlier in our description of the evaluation of the learning component, we have indicated four interaction conditions for knowledge acquisition. In three of the four conditions, the SBF model of the new device needs to be generated, and it is done by method (3).

In addition, the SBF models of new devices may also be acquired directly from Natural Language descriptions of devices. In a companion project called KA (Peterson et al., 1994), such acquisition of SBF models has been modeled. KA can acquire the SBF models similar to the ones used in IDEAL. Moreover, the earlier versions of retrieval and problem-solving modules of IDEAL (i.e., KRITIK2 system) have been used in KA.

9.2.3.3 Use of the Same Representations for a Completely Different Task

The content theory of qualitative device models has been initially developed for the context of design task. But we have subjected this to the constraints of a completely different task. In particular, we refer again to the KA project in which the same SBF representations were used but for a different task. KA's task is understanding of Natural Language descriptions of designs and design problems. This indicates that our theory of qualitative modeling is flexible and general to support multiple functionalities (i.e., tasks).

9.2.4 Memory: Organization, Indexing, and Index Learning

We have evaluated this component of our theory along the following two dimensions:

1. Efficiency of Retrieval for Different Indexing and Organization Schemes
2. Use of the Indexing Scheme for Retrieval in a Completely Different Task

IDEAL uses model-based indexing of design analogues and design patterns. It could learn multiple types of indices, functional and structural, for design analogues automatically as design analogues were acquired and learn functional indices for design patterns as they were acquired. It organized the design analogues in multiple hierarchies. In our evaluation with 20 designs, IDEAL's model-based index learning was found to be more effective in the performance task of retrieval of design analogues than its indexing without the knowledge of models. With the addition of design analogues to memory, the average retrieval time typically increases irrespective of the memory organization, but in IDEAL it increases *slower* if model-based indexing and organization were used than otherwise. IDEAL's model-based indexing and organization are such that the addition of more and more analogues in a given domain affects the retrieval of analogues only within that domain.

9.2.4.1 Efficiency of Retrieval for Different Indexing and Organization Schemes

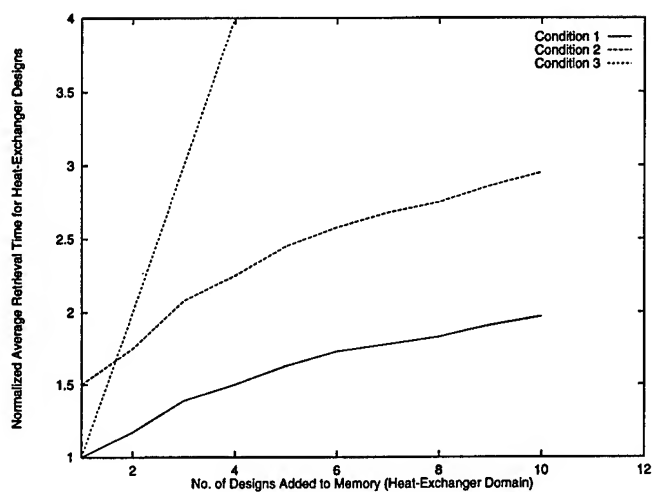
The model-based approach is quite efficient because the organization of analogue memory and the functional indexing scheme enable a quick retrieval of analogues relevant to the current problem. We have measured the effect of model-based indexing on the efficiency of analogue retrieval and the measurements as described below support this conclusion.

We used 20 different designs from two different domains (electric circuits and heat exchangers) to test the effect of model-based index learning on the retrieval of analogues. The independent variable is the number of analogues added to memory and the dependent variable is the normalized average time for retrieving any of the analogues in memory. The retrieval time is measured in terms of the number of comparisons needed between a specified problem and the stored analogues along each dimension (i.e., property of substance in functional specification) common to the problem and the stored analogues. The retrieval time is normalized with respect to the time it takes to retrieve an analogue when only that analogue is in memory.

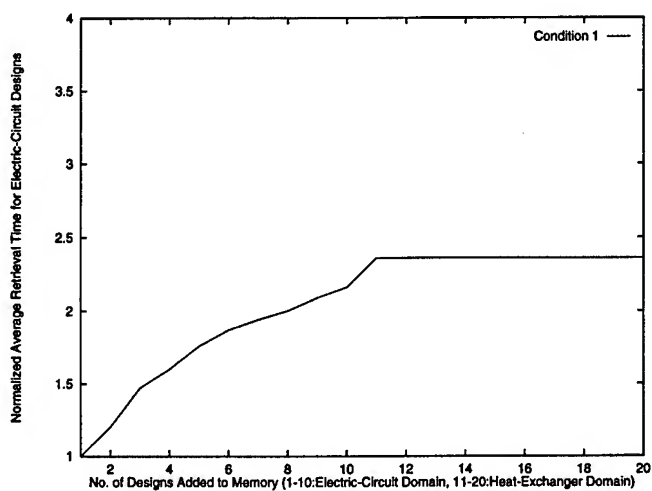
As analogues are added to any memory, the subsequent retrieval time increases. In the first set of experiments, the question was whether model-based index learning has any useful effect on the growth of the retrieval time. Within each domain, we compared the retrieval times on 10 problems as the 10 analogues are added to memory under three different conditions: (1) analogues are stored using model-based index learning, (2) analogues are stored along all possible features in the respective problems (i.e., models are not used to select the relevant indices), and (3) analogues are stored without any organization in memory (i.e., the bottom-line condition in which the retrieval requires an exhaustive search through a list). The results are shown in Figure 9.8(a). It is evident from the graph in Figure 9.8(a) that the rate of growth of retrieval time in the model-based index learning condition is the slowest. The next best is condition (2), with condition (3) being the worst. The reason condition (1) is better than condition (2) is precisely because the SBF models help to store the analogues in a relevant, smaller number of features in the problems. In this experiment, new indices are added to memory when the first analogue is stored. The difference in the number of features selected under conditions (1) and (2) is just one. The difference in the retrieval times in these two conditions when only one analogue is stored hence indicates the advantage due to pruning out merely one feature. In condition (2), the retrieval time grows faster as analogues are added because of the addition of analogues along irrelevant feature(s) which in turn increases the retrieval cost due to matching on those features also. This experiment is controlled such that there is no confounding effect due to retrieval on partial match (because a partial match requires less number of comparisons in a hierarchy, as the search would stop at a higher level, than a perfect match requires). That is, for each of the problems used to measure the retrieval time, there is a perfect match in the stored analogues, and the retrieval under all three conditions results in the retrieval of the same analogues.

We have also tested another effect of model-based indexing on the analogue retrieval. The question here was how the addition of analogues to memory under this indexing scheme in one domain affects the retrieval of analogues in another domain.⁵ We first stored the 10 analogues in the domain of electric circuits. There are two features under which IDEAL stored these analogues hierarchically based on the feature values. Then, we measured the normalized average retrieval time on the retrieval of these 10 analogues as 10 more analogues from the domain of heat exchangers are stored. IDEAL stores the second set of 10 analogues under different features than those for the first 10 (as the different sets of features in the problems characterize the domains to be different). The results are shown in Figure 9.8(b). As evident from the graph in Figure 9.8(b), the retrieval of analogues in the domain of electric circuits is unaffected with the addition of analogues in the domain of heat exchangers except for the spike in the retrieval time once when a new feature is added to memory as an index. The spike in the retrieval time is due to the comparisons required at the root node in analogue memory to discriminate between

⁵Two domains are considered distinct if the structural elements (e.g., batteries, pipes) in the domains are different.



(a) Effect on retrieval under 3 conditions of indexing



(b) Effect of the addition of analogues in one domain on the retrieval in another

Figure 9.8: Performance Measures on the Analogue Retrieval Task

the features (for instance, to select the hierarchy under *voltage* and weed out the hierarchies under *acidity* and *state*). Thus model-based indexing is effective in grouping analogues based on their content, and this in turn enables the retrieval of only semantically relevant analogues for given problems. Although this suggests a useful effect of model-based indexing on the quality of retrieval in restricting the retrieval to a relevant domain, it is yet to be empirically determined how exactly it affects the quality of retrieval within a given domain.

9.2.4.2 Use of the Indexing Scheme for Retrieval in a Completely Different Task

Initially, we developed our theory of indexing and index learning for retrieval in the context of design task. In this task context, the problems are specified in the SBF language and can involve both functional constraints and structural constraints. Later, we have subjected this theory to the constraints of retrieval task in a completely different task context. In particular, we refer to the KA project (Peterson et al., 1994) in which one task is to design for problems specified in Natural Language. Our indexing scheme worked equally well for the retrieval in that context. In fact, the KA project indicated that the structural indexing scheme is essential for covering some classes of design problems because their descriptions in Natural Language may only refer to the structure and none about the functions.

9.3 Limitations

Although our theory covers a number of issues in innovative design in sufficient depth as illustrated in the evaluation, there are some limitations. Like the evaluation of the theory, the limitations can also be stated at two different levels of the theory: the theory as a whole and the individual components of the theory. Unlike the descriptions of the evaluation, we will describe first the limitations of the components of the theory and then the limitations of the theory as a whole.

9.3.1 Limitations Of Components of the Theory

Recall that our theory has four main components: problem solving, learning of design patterns, qualitative modeling and learning of device models, and memory (i.e., its organization, indexing, and index learning). In the following sections, we will describe the limitations of these four components of our theory.

9.3.1.1 Problem Solving

In our theory, the problem-solving component includes the stages of analogical transfer and modification (i.e., spawning of adaptation goals and achieving them), and evaluation of the solution. As we have already mentioned, the coverage of this theory depends on the adaptation strategies that are assumed to be known or that can be learned. Those strategies determine

the classes of design problems that can be solved. Currently, our theory can deal with only a limited number of different types of functional differences, for instance, only substance difference, substance-property-value difference, and substance-property-value-fluctuation difference. More specifically, the differences between the functions of a target design and a source design can be only in the output states (i.e., MAKES states of the functions); however, when the difference is substance-property-value-fluctuation type of difference, then it can be either in the input states of the functions or in the output states. While our theory can deal with differences in components that are due to the substance-related differences, it cannot cover differences in functions that are expressed in terms of component states. Neither can it cover design of devices with fields (for instance, electro-magnetic devices with electric field and magnetic field). Similarly, it is not clear whether our theory is applicable outside the design of physical devices, for instance, design of pictures (or buildings) where spatial reasoning may be more predominant than functional and causal reasoning.

Furthermore, our theory is applicable only when the SBF model of a design to be modified is available. And, in particular, the SBF model needs to specify the qualitative or quantitative relationships between different properties of substances and components. That type of knowledge is needed in the device model in order for the diagnosis to work as well as for the repair and evaluation. If the relationships are qualitative, they can be only directly-proportional-to or inversely-proportional-to. On the other hand, if the relationships are quantitative, they can be simple algebraic equations relating one or more variables that conform to these constraints: if there is only one term in the equation, then it can have multiple variables, but otherwise, each term in the equation can have only one variable.⁶ These assumptions are important because the inferences required in problem solving regarding the changes in values of variables and the inter-dependencies between variables are limited by the assumptions.

9.3.1.2 Learning of Design Patterns

Although our theory of learning of design patterns has been evaluated extensively and has been made to be flexible, there still are some limitations, because the issues it addresses are hard. While the theory covers a reasonable variety of GTMs (in exact, six of them), its coverage of GPPs is poor—learning of only two GPPs has been explored. Even for the specific GTMs, learning has not been explored for some specializations of the GTMs. For instance, the current theory has been tested only for serial cascading but not for parallel cascading. Similarly, there are only four different types of feedback mechanism and only one type of feedforward mechanism for which our theory has been tested.

The current theory only accommodates learning at the time of storing a target analogue. That is, design patterns are learned when a target analogue is available (either generated auto-

⁶An example of the acceptable equation with one term is $x = a_1 * y/z$. An example of the acceptable equation with multiple terms is $x = a_1 * y \pm a_2/z \pm a_3 * p^2$. This analysis has benefited from discussions with Bill Murdock.

matically or given by an oracle upon problem-solving failure) for storage. But, in general, design patterns may very well be acquired at the time of solving a problem in a different domain. A preliminary analysis has indicated that the same learning methods would be useful for learning at the problem-solving time also but different indexing and organization of design analogues would be necessary because the specific design analogues from a different domain would need to be accessed.

Also, the issues in the incremental revision of learned design patterns have not been explored in sufficient depth. It is not clear yet, when and whether a new GTM is altogether formulated in a learning situation, or whether an existing GTM is revised. In the current theory, a new GTM is first formed in each learning situation. Then the new GTM is checked to see if it “merges” with an already acquired (or available) GTM—if so, the available GTM is revised to incorporate the additional content from the new GTM; otherwise, the new GTM is stored as a separate GTM.

In our theory, the learning of design patterns involves comparing the SBF models of two designs, one without an instance of the GTM to be learned and the other with an instance of that GTM. The current theory assumes that the behaviors in both the SBF models are described at the same level of detail. Ideally, the theory should cover the comparison of two behaviors described at two different, arbitrary levels of detail. The theory also assumes that there is only one GTM instantiated in the behavior of the new design from which learning occurs. That is, for instance, the new design example cannot have both the cascading mechanism and the feedback mechanism, at least not in the same, single behavior.

9.3.1.3 Qualitative Modeling and Learning of Device Models

In our theory, new SBF models are acquired by three different methods: (1) revision of known, similar devices, (2) instantiation of design patterns in known models, and (3) a combination of model revision and primitive-behavior composition (given a partial or complete structural specification of the new device). The first method is limited by the kinds of revision strategies that our theory currently accommodates. That is, those revisions are simple and local as determined by the adaptation strategies: substance substitution, substance-property modification, component replacement, and component-parameter modification. Similarly, the second method is limited by the kinds of design patterns the theory accounts for, namely, cascading, feedback, and feedforward GTMs. But these revisions are more complex and non-local, which involve changes to the structure of the known models.

Since the third method involves an additional dimension of processing from structure and composing the behaviors of primitive structural elements, the limitations of the method are characterized by the kinds of input structural specifications relative to the structures of known devices it can handle (i.e., the classes of structural differences for which it can revise the known models). Currently, this method accounts for learning new device models only under the fol-

lowing conditions: (1) the new device structure can have one or more new structural elements compared to those in a known device structure, but there cannot be less; (2) the new structural elements are only serially connected to those that map onto the structural elements in the known device (but can handle circular connections of the structural elements also when the direction of flow of substances through each element is known); (3) the primitive behavior of each new structural element needs to specify the input substance and the output substance (if they are different) and the direction of flow for the substances through the structural element; (4) the primitive behavior of each new structural element needs to also specify the qualitative/quantitative relationships between properties of the input substance and those of the output substance; (5) each new structural element can have only one input and only one output; and (6) the SBF model of the known, similar device needs to represent the internal causal behaviors such that the behavioral states are specified at the locations that are the connecting points between the structural elements.

9.3.1.4 Memory: Organization, Indexing, and Index Learning

In our current theory, all the knowledge is indexed in a task-directed manner. This, of course, implies that the indexing schemes may be limited only to those tasks for which they are designed. While the design analogues are organized hierarchically in multiple ways, the organization of design patterns is poor and only flat. In general, a theory perhaps needs to have a better account of the organization of design patterns. Moreover, the two types of design patterns the theory covers, i.e., GPPs and GTMs, are not currently cross-indexed, and they themselves are not connected to design analogues. But, in general, to enable a wider class of tasks, cross-indexing of the design analogues and design patterns may be necessary.

As pointed out in the description of evaluation of indexing, the theory is limited in its characterization of the quality of retrieval of design analogues. As a result, the effects of particular indexing schemes on retrieval quality are not yet clear.

Our theory of model-based index learning too is limited in that it is applicable only when specific types of knowledge are available in the SBF models of design analogues. For instance, learning of functional indices is better only when the transitions in the SBF models explicitly specify conditions on the properties of substances in the functions. Similarly, learning of structural indices is better only when the transitions in the SBF models explicitly indicate conditions on the properties of components and structural relations among components. Although our theory accounts for how the new SBF models may be acquired in different ways, it does not indicate how these specific types of knowledge in a new SBF model may be acquired.

9.3.2 Limitations Of the Theory as a Whole

Our evaluation at the level of the theory as a whole concerns with the three facets of innovative design, namely, non-local modifications, cross-domain transfer, and problem reformulation. We

will now give the long-deferred description of how well our model-based theory of innovative design accounts for these three facets of innovative design, i.e., we will give for each of the three facets, what classes our theory covers and what it cannot.

9.3.2.1 Coverage of Non-Local Modifications

The question here is what classes of non-local modifications does our theory cover, and how well does it cover them. Since in our theory, the non-local modifications to known designs are primarily done by instantiation of design patterns, the coverage is limited to the kinds of design patterns known (or that can be learned). That is, the design patterns are the GTMs of cascading, feedback, and feedforward. However, there is also an adaptation strategy called “component addition” that can be applied to make non-local modifications by adding a component in the structural topology of known devices. Each of these different GTMs enables a different type of non-local modification, and so does the component-addition strategy. Table 9.2 summarizes the types of non-local modifications that the theory can account for and the limitations.

Table 9.2: The Types of Non-Local Modifications that our Theory can Account for and its Limitations

Design Pattern or Adaptation Strategy	Type of Non-Local Modification	Limitation
Cascading GTM	Serial addition of one or more device elements with the same function into the structural topology of a known device.	Cannot account for Parallel addition of device elements to the known structure.
Feedback GTM	Addition of a causal loop, that is, the addition of a device element in parallel to the known device but with the reverse flow of substances.	Cannot account for Open Loop Feedback, and those types of Closed Loop Feedback that require both a Sensing element and a Comparator element.
Feedforward GTM	Addition of a device element in parallel to the known device with the same flow of substances.	Cannot account for those types of feedforward that require both a Sensing element and a Comparator element.
Component Addition	Serial addition of a single component to the structural topology of a known device.	Cannot account for non-serial and non-parallel additions of components such as inclusion.

A complement to the component-addition strategy is component deletion. Application of component-deletion strategy can also enable non-local modifications to known devices. But IDEAL is limited in that it does not have this strategy and hence it cannot currently make such non-local modifications as in component deletion.

9.3.2.2 Coverage of Cross-Domain Transfer

The question here is, of course, what types of cross-domain transfer does our theory account for, and how well does it account for them. In our theory, cross-domain transfer occurs via design patterns. That is, the design patterns may be learned in one domain and used in another. In fact, the design patterns themselves are the knowledge that gets transferred. Therefore, in that sense, our theory accounts for only abstraction-based cross-domain transfer. Furthermore, it covers such transfer only in the context of design of physical devices where a design analogue consists of the problem specification (i.e., the desired function), its solution (i.e., the structure of the device), and a model of the solution (i.e., an SBF model that explains how the structure delivers the desired function).

Another type of cross-domain transfer may also involve design patterns, but the transfer may occur between two specific designs. In this type of transfer, the design patterns are learned at the time of problem solving (as opposed to the time of storing a new design). Our current theory of analogical design does not cover this type of cross-domain transfer.

Yet another type of cross-domain transfer may not even involve design patterns or any abstractions. Instead, the transfer might occur *directly* between a source analogue and a target problem. However, the applicability of such transfer is not clear for the situations where a source analogue and the target problem are from two distant domains. An example of a theory that covers only direct transfers is Structure-Mapping Theory (Gentner, 1983). Our theory of analogical design currently does not cover this type of cross-domain transfer either.

9.3.2.3 Coverage of Problem Reformulations

The third important issue in our theory of innovative design is problem reformulation. The evaluation question here is what classes of problem reformulations does our theory account for. In our theory, problems are reformulated when the designs fail in new environments. Design patterns, in particular, GPPs, help in understanding design failures by forming causal explanations and in formulating new constraints. Our theory is limited to covering a small class of design failures that can be specified as undesired behavioral state transformations. An example of such a failure is the undesired cooling of hot coffee in a coffee-making device. Moreover, our theory is limited to the class of problem reformulations that involve addition of new functional constraints to the problem (i.e., addition of new sub-functions to the problem specifications); these new constraints are assumed not to interact with the old constraints.

In general, problem reformulations may also involve deletion of constraints from the problem specifications and modification of constraints. But, our theory currently does not cover these types of problem reformulations.

CHAPTER X

RELATED WORK

Our work intersects with several lines of research in AI: qualitative modeling, design, case/model-based learning (of abstractions, strategies, and indices), and analogical reasoning. It builds upon previous work in these areas but also explores new issues in new directions. Here we relate our work to previous work along the following dimensions: tasks addressed in a computational model (i.e., types of design tasks, types of learning tasks, etc.); methods used for each of the tasks (i.e., analogy for design, model-based method for learning, etc.); types of knowledge used and learned (i.e., types of design cases or analogues, device models, design patterns, indices, etc.); inferences and control of processing; domains covered by a computational model (i.e., single domain vs. class of domains); class of problems solved in a domain (i.e., complete generation of designs as in the function-to-structure mapping task, design completion, etc.); and autonomous vs. interactive (i.e., whether the agent is completely autonomous or needs human intervention during the process).

10.1 Qualitative Modeling

AI research in qualitative modeling goes at least as far back as Hayes (1979). Much work on naive physics and qualitative reasoning (e.g., Hayes, 1979; Hayes, 1985; de Kleer and Brown, 1984; Forbus, 1984; Kuipers, 1984; Iwasaki and Simon, 1986) has focused on the representation of the physical world (including devices) and the simulation of causal processes. Similarly, some work in human-machine systems (e.g., Govindaraj, 1987; Rasmussen, 1985, 1987) has dealt with the representation of plant operators' mental models of large, dynamic, physical systems. Yet other work in psychology has also considered how models capture other types of relations such as structural relations between electrons and the nucleus in an atom (Gentner, 1983). Our work differs from the past work in qualitative modeling in a number of different dimensions: (1) content and representation, (2) indexing and organization, and (3) acquisition of particular kinds of models. An important difference between our work and most previous work in qualitative modeling is in the organization of models: we emphasize hierarchical and functional organization of behaviors. In particular, our work emphasizes that the way device models are organized in memory and linked to other types of knowledge significantly affects their use in various problem-solving tasks and their acquisition.

10.1.1 Content and Representation of Device Models

Goel's SBF models, Sembugamoorthy and Chandrasekaran's Functional Representation, and Bylander's Consolidation: Our work borrows from KRITIK (Goel, 1989) the representation of device models in the form of SBF models and extends it to represent different types of abstract, generic models of design knowledge—models of design patterns such as GPPs and GTMs. IDEAL's SBF representations are based on the same component-substance ontology as developed by Bylander and Chandrasekaran (1985). The difference is in the methods of acquisition as we describe later as well as the internal organization of device models. That is, the SBF models organize the internal causal behaviors hierarchically based on functions of substructures. Both IDEAL and KRITIK share Sembugamoorthy and Chandrasekaran's (1986) notion of an explicit representation of internal causal behaviors and their indexing by functions. That is, the SBF models share the principles of functional representation framework (Chandrasekaran et al., 1993). But Sembugamoorthy and Chandrasekaran's work does not use a well-defined ontology for behaviors. In fact, SBF models combine the good aspects of functional representation scheme and component-substance ontology.

Rieger's Commonsense Algorithms: Rieger's work (Rieger, 1975; Rieger and Grinberg, 1978) on commonsense algorithmic representations emphasizes the explicit representation of causal knowledge of how to do things and how things (e.g., physical devices) work. In his work, the internal behaviors are represented as compiled sequences of behavioral state transitions. The SBF representations are similar to that in capturing causal knowledge explicitly in sequences of behavioral state transitions. Like in Rieger's work, a behavioral state in SBF models too is represented in terms of state variables describing the partial state of a device. While the representation of transitions in SBF models is also motivated by the commonsense representations of Rieger, the specific types of structural and causal relations that annotate the transitions are different. The differences are partly due to the difference in the use of the representations (for instance, design vs. language comprehension).

Forbus' Qualitative Process Theory: Our work concerning models of devices and physical processes is related and similar to Forbus' work (1984) on modeling physical processes, yet quite distinct from it. In Qualitative Process Theory (QPT), Forbus provides a formalization of process that can be used in qualitative dynamics. In QPT, the explicit representation of a process is for a generic physical situation, and it captures all possible behaviors. In that sense, the representation of a physical process in an SBF model of a device is different as it captures a specific process. However, our representations of GPPs are similar to the representations of processes in QPT, although the specific content theory is different. That is, our theory has representational analogues to constructs that Forbus has proposed. For instance, the "quantity conditions" and "relations" in the representation of a process in QPT capture a similar type of knowledge as do our representational primitives "under-condition-substance" (i.e., behavioral requirements on the properties of substances) and "parameter-relations." Although QPT distinguishes be-

tween relations and influences, our representations do not make such a distinction. Instead, our representation explicitly captures the individual state changes and sequences of state changes, and causes for those changes, which immediately indicates what relations are important in a particular state change. In QPT, processes need to be applied to specific instances of individuals to generate behaviors, while they are explicitly represented in SBF models. IDEAL's representations are different partly because it takes an adaptive approach to modeling devices while QPT takes a compositional, generative approach. And, the differences are also due to the tasks IDEAL addresses being different from those QPT does. Further important differences between QPT and SBF models are that the former does not have any explicit notion of function and it does not capture quantitative relationships. In the SBF representations of GPPs a function is a behavioral abstraction.

deKleer's Qualitative Differential Equations: Similarly, our work on modeling devices with feedback control is well related to deKleer's (1984) work on representations of circuits with feedback using qualitative differential equations and confluences. (His system was called EQUAL.) But there are a number of differences on several aspects in the two representations, which we believe are important for enabling certain classes of design problems. First, IDEAL not only represents the causal processes in specific devices with feedback but also (acquires and) represents them in generic terms. IDEAL's representation of feedback enables it to solve design problems that involve devices with feedback and enable cross-domain transfer of that knowledge. While IDEAL's representations explicitly capture the global causal pattern in the behavior of a device with feedback, deKleer's representations of the feedback only specify the relationships between individual variables. Furthermore, deKleer's representations do not capture the notion of fluctuations in the input/output values, although we believe that if the representations of feedback were to support design of devices with feedback, they need to capture the notion of fluctuations explicitly.

Govindaraj's Qualitative Approximation Methodology: KRITIK's primitives for functions of components, and hence those of IDEAL, are related to but not the same as the primitives proposed in Qualitative Approximation Methodology (Govindaraj, 1987). Govindaraj, in his methodology to model large dynamic physical systems, has proposed primitives to describe components in a system; but those primitives are *primitive components* and not primitive functions. However, the primitive components are closely related to "primitive" functions as they are responsible for the basic functions performed by the components. For example, the primitive *conduit* in Govindaraj's set of primitives has a function similar to the primitive function *allow*. He developed the representations in the context of simulation of physical systems and tutoring the operators of the systems, both very different from the design task. He has applied the methodology to model very large systems, consisting of the order of 500 components, but IDEAL's representations are not yet shown to be scalable to that level of complexity. The hierarchical organization of the physical system in his work follows the principles of Rasmussen's (1985) hierarchical knowledge representation (and representation at multiple levels of abstraction), and

is also similar to our functional organization of behaviors.

Gero et al.'s Function-Behavior-Structure Models: IDEAL's SBF models are also related to the function, behavior, and structure representations of design knowledge and design prototypes developed by Gero and his colleagues (Gero, 1990; Gero et al., 1991). However, their representations are generalized models as opposed to specific SBF models of devices in IDEAL. While in their work, design prototypes are intended to capture generalized design knowledge, in ours, GPPs and GTMs capture two very different kinds of generalized knowledge, one about devices and the other about design strategies. In contrast to IDEAL, their representation of behaviors is only in terms of variables linked to structural elements. In fact, by behavior, Gero refers to the output behavior of an artifact/device, while behavior in IDEAL's representations refers to the internal causal behavior of a device. As a result, in IDEAL, the representation of behavior explicitly captures the states, state changes and their causes, and thus provides a functional organization for behavior variables.

10.1.2 Content and Representation of Generic Models

Our work on modeling design patterns such as GPPs and GTMs is related to previous work on *generic models* as in modeling of qualitative processes (Forbus, 1984), modeling of causal patterns (Pazzani, 1991), and modeling of scientific discovery (Darden, 1991; Nersessian, 1995a). As we have already discussed in the previous section, GPPs are similar to Qualitative Processes yet different in the specific content theory. Also, Forbus' work does not cover modeling of generic mechanisms.

IDEAL's causal patterns, namely, the generic models of physical processes, are similar to the general patterns of causality used in OCCAM (Pazzani, 1991). However, the specific content of the patterns is different because their domains are very distinct. While IDEAL's domain is physical devices, OCCAM dealt with the patterns of causality in social domain (i.e., volitional agents' goals, actions, and events in the context of story understanding). AQUA (Ram, 1989) also has patterns of causality in social domains represented as explanation patterns that it uses for story understanding. OCCAM cannot learn its causal patterns while IDEAL and AQUA can learn their respective patterns from experiences. In addition, IDEAL also has another type of design patterns, namely, the generic models of engineering mechanisms (i.e., GTMs).

IDEAL's generic models of engineering/physical mechanisms are also similar to some of the concepts postulated in some recent work in scientific discovery. For instance, Darden's (1990) work on modeling Mendelian's theory of genetics has found models to be useful in theory revision or redesign; and Nersessian's (1995a, 1995b) work on analyzing historical records of Maxwell's discovery of electromagnetic field equations suggests that Maxwell used generic models in his reasoning by analogies. In addition to generic mechanisms, our theory also covers the content, representation and acquisition of other types of generic models such as physical principles and processes. We have also shown how those other types of generic models facilitate discovery of

new constraints in a problem and thereby support innovative adaptive design (Bhatta, Goel and Prabhakar, 1994).

Darden's work views scientific theories as devices and theory revision as design-adaptation task. In some personal communication (Darden, 1991), she has conjectured that the content theory of GTMs and their use for design adaptation might provide a basis for modeling the formation of early theories of heredity and genetics. Thus the use of GTMs could be a very general domain-independent innovative method of case adaptation.

Similarly, Nersessian's (1995a, 1995b) analysis of Maxwell's discovery of electromagnetic field equations starting with the theories of Newtonian or continuum mechanics suggests the use of generic models or abstracted models in the theory formation. However, Maxwell's representations of those models were often diagrammatic or visual. But, nevertheless, the kinds of relationships captured in those models are similar to the kinds of relationships in the generic models of physical processes such as the process of heatflow (and generic models of physical principles such as the zeroth law of thermodynamics), and the generic models of the engineering mechanisms such as cascading.

Generic Tasks: Since IDEAL's knowledge of generic teleological mechanisms is strategic knowledge for solving some classes of design adaptation tasks, the models of GTMs seem similar to Chandrasekaran's Generic Tasks (Chandrasekaran, 1983) for any problem solving in general. Chandrasekaran analyzed that all problem solving tasks may be classified in terms of generic tasks. The advantage of such a classification, he suggested, is that the generic tasks can then become building blocks for systems to solve any problem-solving task in specific domains. Then an arbitrary problem-solving task can be solved by a combination of the methods/strategies in generic tasks. Similarly, GTMs are such "generic" tasks for some classes of design adaptation. That is, given a design adaptation task, we believe, it can be solved by applying one or more of GTMs. Like generic tasks, GTMs specify the tasks—the task is specified in terms of the functional difference a GTM reduces. Like generic tasks are strategies for tasks, GTMs too are strategies for design adaptation tasks—the strategy is specified in terms of the patterned modification a GTM suggests to a behavior in order to achieve an instance of the task it represents.

AUTOGNOSTIC (Stroulia, 1994) is a model-based reflective system that can learn new knowledge and new strategies for problem solving from failures. It uses models represented in a similar language as IDEAL's (i.e., SBF representations) for capturing the functioning of a problem solver, which is an abstract device. In addition, it uses models of generic tasks that enable AUTOGNOSTIC to redesign the problem solver itself to include new subtasks. That in turn enables it to expand the class of problems it can solve. Similarly, IDEAL's models of GTMs enable it to expand the class of design adaptation tasks it can solve, although it does not do a reorganization of its own task structure. (We will compare AUTOGNOSTIC with IDEAL on other dimensions also in later sections.)

10.1.3 Indexing and Organization

Most previous work on qualitative modeling has ignored the issues of indexing models and organizing them in memory. In fact, none of them (perhaps with one or two exceptions that are direct precedents of this work) advocate explicit, declarative representations of models, let alone their indexing and organization. But we consider those issues as important as the issues of representation, because we take a memory-based view of qualitative modeling. IDEAL's indexing of causal behaviors in SBF models by the device functions is similar to Sembugamoorthy and Chandrasekaran's (1986) scheme. But like KRITIK, IDEAL also organizes the causal behaviors hierarchically both by functions and structure. In our theory, the cross indexing among several different types of knowledge that constitute SBF models is considered very important because such indexing not only enables a number of inferences but enables them efficiently too. For instance, in IDEAL, (1) functions index into internal causal behaviors, (2) behaviors themselves index into functions at lower level of structural detail (i.e., substructures), (3) behaviors also index into structural elements and relations among the elements, and (4) behaviors further index into other behaviors in a device (i.e., for instance, temporal simultaneity of two causal behaviors is captured by representing mutual dependency between them). Thus in IDEAL the internals of a model are hierarchically organized. A few examples of the inferences where the cross-indexing and organization of models play a role are retrieval of a known model, localization of needed modifications in a model, and localization of simulation of the model. Such inferences are much more computationally expensive for modeling of physical devices if the representations were as in most qualitative physics work (e.g., de Kleer and Brown, 1984; Forbus, 1984; Kuipers, 1984; Iwasaki and Simon, 1986).

Similar to IDEAL, AUTOGNOSTIC (Stroulia, 1994) also considers the indexing and organization of SBF models serious and exploits those aspects in its reflective reasoning and learning from failures. AUTOGNOSTIC organizes its SBF model of the problem solver hierarchically and indexes the model by a functional characterization of the information-processing task its problem solver solves.

Our memory-based view is closely related to other work on memory-based approaches, for example, (Minsky, 1975), and especially (Schank, 1982) and (Kolodner, 1984), although these are not in the context of qualitative modeling. In *Dynamic Memory* Schank (1982) proposed Memory Organization Packets (MOPs) for organizing certain kinds of information, for example, the goals of volitional actors, and the sequences of actions performed to achieve the goals. He described how MOPs can facilitate story interpretation and enable generalization and learning from past experiences. We adopt a similar view towards qualitative models—that is, device models, represented in the form of SBF models, organize teleological, causal, and structural information underlying the functioning of devices. They facilitate such tasks as interpretation of design descriptions, designing of new artifacts, and learning of design patterns from design experiences.

10.1.4 Acquisition

Although most previous work on models of physical phenomena has concentrated on their representation and use in reasoning, there is some previous work on their learning as well. For instance, KRITIK itself addresses some forms of model acquisition.

KRITIK: In KRITIK, new models are acquired by revising the models of known similar devices (Goel, 1991b). IDEAL inherits the same method of model revision from KRITIK. But IDEAL also has two additional methods of acquiring new models: (1) a combination of the methods of model revision and primitive-behavior composition and (2) instantiation of design patterns in known models. Both these new methods in IDEAL enable non-local, patterned modifications to known models (i.e., changes to the topology of the causal structure in internal behaviors of known devices) while KRITIK's model revision method is only capable of simple, local modifications (such as substance substitution and component replacement). IDEAL's combined method (i.e., model-revision + primitive-behavior composition) is capable of non-local modifications to known models because it can reason about new device structures that have new components (relative to a known device structure) connected to the other components in different structural relations: in series, in parallel with same substance flow, and in parallel with opposite substance flow.¹ Of course, that is possible only when the behaviors of new components are known and the direction of substance flow through each new component is also known. IDEAL's second method that uses the knowledge of design patterns is capable of non-local modifications precisely because the design patterns encapsulate when and how to make such modifications to known models.

Although both KRITIK and IDEAL build on the same component-substance ontology developed by Bylander and Chandrasekaran (1985), they both share a critical difference with the latter on model acquisition. That is, in Bylander and Chandrasekaran's work, the internal causal behaviors of devices are constructed at runtime from primitive behaviors by consolidation, while in both the former systems, they are generated by revising the models of similar known devices.

Learning of Qualitative Processes: Forbus and Gentner's (1986) work on learning of Qualitative Processes (QPs) is especially relevant to the task of learning GPPs in this research. They are similar in that the output of the learning task is the same type of knowledge, namely, physical processes such as heat flow. But the inputs to the learning tasks are quite different, and so are the learning methods. In Forbus and Gentner's work, the task is to learn a specific QP from a given specific physical situation (such as an observation that the temperature of some object decreases) in a new domain by mapping the explanation of a similar QP (such as liquid flow) in a different domain. However, in our work, the task involves learning a generic description of a physical process (such as the Heat-Flow GPP) from two specific design examples involving different instances of the GPP by abstraction over the similarities and differences in the specific models of the designs.

PHINEAS: Similarly, Falkenhainer's work (1990) on learning of theories of physical behaviors

¹The parallel connection with opposite substance flow creates a cycle of substance flow in the device.

in his PHINEAS system is also related to the tasks of learning of specific SBF models of devices and GPPs. However, similar to Forbus and Gentner's work, the task that Falkenhainer addresses involves learning of a theory of a specific physical behavior by a similarity-driven explanation mechanism. This mechanism involves analogical mapping of the explanation of a similar physical situation. Keeping aside the differences in the representations, this task is similar to IDEAL's task of learning a specific SBF model of a device from that of a similar device. But, IDEAL goes beyond this in that it also addresses the task of learning abstractions such as GPPs.

Physics101: IDEAL also has some similarities to Shavlik's (1985) Physics101 system with respect to learning of physical processes and principles. Physics101's learning task is to form new formulas (in some sense, principles) for physics concepts such as momentum as output, given a single solved example such as that of a 2-body collision problem as input. In contrast, IDEAL's learning task takes as input two design examples of specific devices and produces as output physical processes that are not just formulas. But rather, the output of IDEAL's learning task captures relationships between internal causal behaviors of devices and their output behaviors. The methods that Physics101 and IDEAL use for these similar tasks are also different. While Physics101 uses EBL method along with in-built bias to generalize a specific formula for 2 objects to N objects, IDEAL uses a combination of model-based method and similarity-based method to abstract over specific processes in the two given designs. Since the methods are different, the knowledge they both assume are also different. For instance, unlike Physics101, IDEAL does not assume any in-built bias as to what to generalize but instead it uses the knowledge of functions and differences between the two input designs to guide the process of determining what to learn. Furthermore, Physics101 assumes the knowledge of Newton's laws and calculus to form explanations of the solutions and to rewrite formulas. Since IDEAL's inferences are different (i.e., it does not need to generate explanations from scratch, nor does it need to rewrite formulas), it does not require the knowledge similar to physics laws or calculus. The interaction conditions under which Physics101 learns are limited compared to those IDEAL can handle. That is, Physics101 acquires its input to the learning task when it tries to solve a given problem and fails at it (i.e., applying any of the known knowledge does not lead to finding a value for the desired, unknown variable). While Physics101 can handle only this interaction condition, IDEAL, in addition to this, can handle three other conditions (as we have described in Chapter 9). The interaction conditions in IDEAL differ among themselves with respect to the information presented as input to the learning task.

BAGGER: IDEAL's task of learning the cascading mechanism is very related to Shavlik's later work (Shavlik and DeJong, 1987) on learning rules generalized to N . Shavlik's BAGGER system learns generalized rules that capture repeated applications of a rule from a single example. This is similar to IDEAL's task of learning the cascading mechanism when it solves a design problem by repeatedly applying the component-addition strategy. Although Shavlik and DeJong's work and our work apparently look similar as both tasks involve generalization to N , there are some interesting differences in both the input and the output knowledge structures of the learning

tasks addressed. For instance, IDEAL's learning task takes as input two design examples, one without an instance of the cascading mechanism and the other with an instance, while BAGGER takes only a single example as input. The methods and hence the knowledge that BAGGER and IDEAL use are also different. Like Shavlik's earlier system Physics101, BAGGER also uses EBL method with an in-built bias to generalize to N ; it explicitly looks for rules in a plan that match the syntactic form of an extended rule (focus rule). In contrast, IDEAL does not need such knowledge because it learns from a comparative analysis of two design examples, and instead, uses the knowledge of function and behavior in the SBF models of the devices. Thus IDEAL uses a combination of model-based method and similarity-based method.

While IDEAL learns abstractions from specific designs, BAGGER learns rules that are more specific than the focus rule (that is, the learned rule is generalized to N , but contains specific information from the explanation of the specific example such as objects and actions). In contrast, IDEAL's output knowledge structure such as the cascading mechanism is not only a generalization to N replications of a component/device but also a domain-independent, and hence generic, description of the concept. In addition, IDEAL learns other types of strategic concepts such as feedback and feedforward mechanisms that are not in the class of concepts BAGGER learns. Furthermore, our method involves much more than simple generalization, namely, abstracting relationships between functions and behaviors. Finally, the interaction conditions for BAGGER are same as those for Physics101, and hence are much different and limited compared to those for IDEAL.

10.2 Design

Tong and Sriram (1992) have compiled an extensive list of design systems and classified them along different dimensions such as the types of design tasks they perform, the methods they use, and so on. We basically follow the same classification and compare our system IDEAL (or the process of model-based analogy) with related systems.

10.2.1 Adaptive Design

Research on design problem solving has led to the development of a number of design methods ranging from heuristic association (McDermott, 1982), to constraint satisfaction (Sussman and Steele, 1980), to plan instantiation (Mittal and Araya, 1992; Mittal et al., 1986; Brown and Chandrasekaran, 1989), to reasoning from first principles (Williams, 1991). For instance, R1 (later called XCON), one of the earliest design systems, used forward chaining over rules for configuring computer systems (McDermott, 1982). AIR-CYL (Brown and Chandrasekaran, 1989) and PRIDE (Mittal et al., 1986) designed mechanical devices by selecting and instantiating plans and procedures. All these systems used only a single method of reasoning, and did not reuse the designs they created. Instead, they used a synthetic approach. Further, they performed only very routine design.

The case-based (adaptive) approach is fundamentally different from these synthetic methods. Although the synthetic methods may (and do) play an important role in design adaptation, the adaptive approach views design problem solving in terms of "design evolution" in which new designs are created by modifying, perhaps combining, earlier designs. For the most part, previous work on adaptive design has generally followed the two main computational models of case-based reasoning, namely, transformational approach and derivational approach. Thus, some case-based design systems (e.g., Barletta and Mark, 1988a; Dyer et al., 1986; Hinrichs, 1992; Navinchandra, 1991; Maher and Zhao, 1987) generally follow the first computational model of case-based reasoning (Kolodner and Simpson, 1984; Riesbeck and Schank, 1989; Kolodner, 1993) in which the solutions to previous, similar problems are transferred and "tweaked" to solve new problems. While some others (e.g., Kambhampati, 1993; Mostow, 1989) closely follow the second computational model (Carbonell, 1983) in which the derivational trace by which the previous, similar problems were solved is "replayed" and "tweaked" to solve new problems.

Although the initial work in adaptive design has only focused on routine design and adaptive methods, much of the recent work has attempted to model innovative design as well as routine design, and addressed issues such as design reuse and integration of multiple sources of knowledge and multiple types of reasoning. For example, KRITIK (Goel, 1989; 1991) integrates case-based reasoning and model-based reasoning; CADET (Sycara and Navinchandra, 1992; Navinchandra et al., 1991) also integrates case-based reasoning and behavior-preserving problem-transformation techniques (that enable cross-contextual design retrieval and synthesis of cases); JULIA (Hinrichs, 1991; 1988) integrates case-based reasoning and constraint posting; BOGART (Mostow et al., 1992) uses design reuse via replay of design plans for supporting innovative design; ARGO (Huhns and Acosta, 1992) also uses previous design plans and macrorules, which are learned and stored at different levels of abstraction; and ALADIN (Rychener et al., 1992) uses multiple knowledge sources, such as cases, rules, mathematical models and statistical methods.

Our approach to adaptive design falls under the transformational approach to CBR, but extends it by integrating it with model-based reasoning as in KRITIK. In particular, our work borrows from KRITIK the ontology and representation of device comprehension, that is, SBF models. However, our work goes much further in integrating learning of abstract models with analogical design and explores different types of design tasks, ranging from routine to innovative to creative. Also, IDEAL's theory covers non-local adaptations in design while KRITIK is limited to simple, local adaptations. Furthermore, our work also includes developing behavior-function models, although using the same language as that of SBF models, for representing design patterns.

In addition, while KRITIK deals with purely function-to-structure mapping design tasks, our work accommodates design problems that include structural constraints besides functional ones. Like KRITIK, our work also considers design in the domain of physical devices.

AUTOGNOSTIC (Stroulia, 1994) shares some common themes with IDEAL in the dimen-

sion of adaptive design. While IDEAL's task is design of physical devices, AUTOGNOSTIC does design of problem solvers such as a route-planning system or a device-design system. AUTOGNOSTIC views a problem solver as an abstract device and views its learning from failures as a redesign task. Thus, while AUTOGNOSTIC redesigns abstract devices, IDEAL redesigns physical devices. Both view their respective candidate designs as failed designs and use SBF models of those designs to diagnose possible faults. That is, both do a similar model-based blame assignment. One crucial difference is that AUTOGNOSTIC uses a trace of the problem-solving episode in addition to the SBF model of the problem solver in its diagnosis while there is no equivalent of that in IDEAL's world. AUTOGNOSTIC needs such a trace because its task is to redesign the problem solver itself. Some of the repairs these two systems perform to their respective designs are also interestingly similar. As we discussed in the context of generic models earlier, AUTOGNOSTIC's knowledge of generic tasks is similar to IDEAL's knowledge of GTMs. Furthermore, their use of the respective generic knowledge is also similar: AUTOGNOSTIC uses its knowledge of generic tasks and instantiates a generic task in its failed design to reorganize the task structure, while IDEAL uses its knowledge of GTMs and instantiates a GTM in its failed design to reorganize the device structure. An important difference, however, is that IDEAL can learn its knowledge of GTMs while AUTOGNOSTIC assumes its knowledge of generic tasks.

10.2.2 Analogical Design

Most of the existing design systems that use analogies perform only within-domain analogies. A few design systems that fall into this category are KRITIK (Goel, 1989), CADET (Sycara and Navinchandra, 1992), BOGART (Mostow *et al.*, 1989; 1992), ARGO (Huhns and Acosta, 1988; 1992), STRUPLE (Maher and Zhao, 1987) and JULIA (Hinrichs, 1991; 1988). All these systems operate in the domain of engineering design with the exception of JULIA which is in the domain of meal planning: JULIA designs menus of meals that satisfy multiple, interacting constraints.

KRITIK solves design problems in multiple domains but can transfer knowledge only between problems within the same domain. CADET is a case-based design tool that provides retrieval and adaptation techniques for combining parts of cases in mechanical engineering domain. In solving a large design problem, it can retrieve parts of cases from different domains to solve subproblems and combine them to solve the given problem. However, given a subproblem, it can only retrieve a case from the domain of the subproblem.

BOGART is a partially autonomous design system that helps students learn design process in the domain of digital circuit design. Although it has been tested in different domains, it can only "replay" previous design plans (that are very specific to a domain) in solving problems from the same domain. ARGO also uses previous design plans to solve design problems in VLSI digital circuit design. Although it can also do only within-domain analogies it can transfer design plans

between far-apart problems in the same domain. ARGO's abstract plans are similar in terms of the level of abstraction to our prototype-device models. STRUPLE uses within-domain analogies to solve design problems from the domain of building design. More recently, the architects of STRUPLE (Zhao and Maher, 1992) have proposed a scheme for creative design by analogy and mutation.

Very few design systems perform cross-domain analogies. In CADET, behavior-preserving analogy (Sycara and Navinchandra, 1991) involves using an abstract description of the desired behavior as an index (or its transformation) to retrieve a relevant case from a different design domain and mapping the specific case thus retrieved to the target problem. Our approach involves using the desired function to retrieve an abstract model and solving the design problem by instantiating the retrieved abstract model in the target domain. More importantly, behavior-preserving analogy assumes *a priori* knowledge of high-level abstractions while our approach learns them from design experience.

DSSUA (Qian and Gero, 1992) is a design support system that uses design prototypes to capture design knowledge and performs analogical mappings between designs from different domains. It retrieves a source analogue by analyzing the given design requirements and using any combination of the function, behavior, structure, or operation requirements. It finds mapping to the target problem through causal links represented in design prototype retrieved as source analogue. The mapping process is domain-independent but causation-type dependent. Its mapping process is syntactic like in structure-mapping theory (Gentner, 1983) and involves matching behavior graphs to establish correspondences between the source and target domains. For complex design problems, behavior graphs tend to be large and finding graph isomorphism is NP-complete! Our approach avoids this problem because mapping in our theory simplifies to instantiating a retrieved abstraction to the target problem.

10.2.3 Innovation and Creativity in Design

Although, as mentioned earlier, most early design systems addressed issues in only routine design, there have been some attempts recently towards modeling innovative and creative design. Some of the work mentioned in the context of adaptive design and analogical design has also looked at the issues of innovation. For instance, Hinrichs in his work on JULIA (1991) has explored how case-based reasoning can be integrated with constraint posting to solve design problems in open worlds, where the innovation arises due to the ability to deal with interactions among multiple constraints. JULIA's domain is meal planning and its task is to design menus of meals. Although JULIA's domain is much different from IDEAL's, they both share in the characterization of innovation—at least along the dimension of problem reformulation. However, the problem reformulations in JULIA and IDEAL are driven by different sources and they use different methods. That is, while in JULIA the problem reformulation is due to the late arrival of constraints in the problem specification, in IDEAL it is evaluation-based. JULIA uses constraint

posting and constraint relaxation methods to incorporate late specifications of constraints and take them into account in its design process (which is hierarchical refinement of a general design). In contrast, IDEAL reformulates design problems by discovering new constraints; the new constraints arise from the causal explanations of design failures that are identified in an external evaluation.

Some of the recent work towards innovative design also involved developing exploratory models. For instance, DONTE (Tong, 1992) uses exploration and problem-reformulation techniques in planning the design process for innovative design. In a similar vein, we have looked at the issue of problem reformulation driven by performance of the designs in real environments and considered how that leads to innovation in design (Prabhakar and Goel, 1992; Bhatta, Goel and Prabhakar, 1994).²

Williams (1991) describes an approach called interaction-based design in which new designs are created by reasoning from first principles. This approach views design as a process of building interaction topologies—networks of qualitative interactions between quantities, used by a device to achieve its desired behavior. It also proposes a search method for focusing the design of novel devices among the large space of possibilities.

Navinchandra's (1991) work is also another example of these exploratory models, but it evolves from work on case-based design. In his work on CYCLOPS, Navinchandra (1991) used a case-based approach to architectural design in which new design ideas are generated by exploring the memory for appropriate cases. In this, new designs are generated by composing subcases from multiple cases. New constraints are discovered through the process of demand posting where a problem with the present case is posted to the case base, requesting the previous cases to solve the problem. If a matching case is not found, then the causes of the problem are retrieved and new demands are posted. CYCLOPS shares IDEAL's view of the importance of discovery of new constraints in innovative design.

Some other recent work in design has even attempted to model creative design using such methods as analogy and mutation. For example, Zhao and Maher (1992) have proposed the COMBINE operator that purports to do both analogy and mutation for creative design. They retrieve design prototypes from a case memory in the domain of building design and then apply mutation operators (e.g., COMBINE) to change the retrieved prototype to include portions of other designs. Qian and Gero (1992) have proposed a method for between-domain analogy that uses design prototypes for creative design, and have implemented it in a system called DSSUA.

More recently, Wills and Kolodner (1994) have also looked at such issues in creativity with in the context of case-based design as situation assessment, evaluation, and problem redefinition. Their model called IMPROVISER is based on their observations of designers solving a mechanical design problem. Our work on IDEAL shares the central theme of their work which is that

²Most recently, in collaboration with Prabhakar, we have begun to apply IDEAL's theory to cover the adaptive modeling of environments and devices alike, and consider how their interactions lead to design innovation in complex domains (Prabhakar, Goel and Bhatta, 1995).

past design experiences and memory processes play an important role in facilitating processes of creative design. While there are some common hypotheses in our work such as that the evaluation and problem reformulation are some of the key processes of creativity in design, the specific computational processes in IDEAL and IMPROVISER are quite different. For instance, in IDEAL, these processes are enabled by not only past design experiences but also device models and design patterns. In addition, IDEAL addresses the issue of learning the knowledge it relies on for use in the other processes. In contrast to IMPROVISER, IDEAL covers both internal evaluation of designs and their external evaluation, but covers problem reformulation only due to feedback from external evaluation. While both computational models incorporate flexible control of processes, the specific ways by which they are achieved are very different: IMPROVISER uses an opportunistic model of reasoning (in a blackboard-style architecture) while IDEAL uses multiple strategies and iterative flow of control (with a possibility to explore alternatives).

10.3 Learning

We organize our discussion of related work in learning into the following different parts: (1) index learning (covering Kolodner's, Hammond's, Ram's, and Barletta and Mark's work); (2) learning abstractions from experience (covering Wisniewski and Medin's, Winston's, Shinn's, and Roverso et al.'s work); (3) learning of strategies (covering Waterman's, Mitchell et al.'s, and Birnbaum and Collins' work); (4) model-based learning (covering Winston's, Kedar-Cabelli's, Mitchell et al.'s, DeJong and Mooney's, and Dietterich's work); (5) multistrategy learning (covering Pazzani's, Stroulia's, and Cox's work); (6) multistrategy reasoning (covering Goel et al.'s work); and (7) learning by discovery (covering Langley et al.'s, Falkenhainer's, Rajamoney's, and Karp's work).

10.3.1 Index Learning

Although the issue of indexing has been for long central to AI theories that emphasize the role of memory in reasoning (e.g., Kolodner, 1984; Lebowitz, 1983; Schank, 1982), there are relatively very few models that address the issue of *automatic acquisition* of indices to learned pieces of knowledge. CYRUS (Kolodner, 1984) was one of the earliest systems that addressed the selection of indices to a new episode. It used primarily the discriminability and the previous predictive power of features to select a subset of features as indices. However, it could only choose indices from the predetermined context-dependent feature lists and could not learn the indexing vocabulary itself.

CHEF (Hammond, 1989) uses explanation-based techniques to learn indices to plans from planning failures; in particular, it learns features in its domain that are predictive of negative interactions between plan steps. By learning predictive features, it can select plans that avoid problems. It also indexes plans by the goals that the plans satisfy.

AQUA (Ram, 1989; 1993) uses a *content-based* approach to index learning and uses explanation-based methods within this approach. AQUA learns particular kinds of knowledge that make good indices to explanatory cases by selecting the indices from the predetermined classes of stereotypical concept descriptions that constitute good indices in its domain. But it cannot learn new classes of indices; it can only learn new indices within the known classes.

Our earlier work (Bhatta and Ram, 1991) has also addressed the issue of learning indices to scripts and causal schemas such as XPs (Schank, 1986; Ram, 1990) in the domain of story understanding. In that context, we developed two methods that enable index learning while processing a story, more specifically, while accessing a schema: either by deferring the selection until necessary cues are found in the story, or by inferring necessary indices from the story, or both. The inferencing is done by explaining the given story using any available domain knowledge; the results of the inferencing that enable retrieval of a schema are hence learned as new indices to the schema. We believe that similar methods might be needed for learning new indices to existing cases during retrieval in model-based analogy. Broadly speaking, our current work shares the same assumptions—indexing schemes and methods of index learning depend on the functional requirements of tasks, if not specific to a task—as in (Bhatta and Ram, 1991).

AUTOGNOSTIC (Stroulia, 1994) also performs knowledge reorganization as part of its redesign of a failed problem solver. It uses the SBF model of its problem solver to determine how to reorganize its knowledge and can learn new indices for it. IDEAL's index learning task is similar but not the same. IDEAL also uses the knowledge of SBF models but it learns indices for a different type of knowledge, i.e., design analogues. AUTOGNOSTIC does not have cases of its own problem solving. Instead, the knowledge types for which it may learn indices depend on the specific problem solver it monitors and represents. Furthermore, IDEAL not only uses a model-based method but also integrates it with a similarity-based approach for generalizing its potentially new indices for analogues.

Another example of similar work is Barletta and Mark's (1988) explanation-based indexing (EBI), which provides an account of how explanations can be used to select a subset of predetermined indexing features to index diagnostic cases from the domain of assembly-line manufacturing. Hammond and Hurwitz (1988) also use a similar approach for the selection of features as indices to diagnostic failures (i.e., failures in diagnosis). We use explanations in the form of SBF models to learn indices (both new indexing vocabulary and right level of generalization) to not only design analogues but to learned abstractions as well. EBI does not account for learning new indexing vocabulary. While both Barletta and Mark's work and Hammond and Hurwitz's work address only learning of one type of indices, our work accounts for learning of multiple types of indices, i.e., IDEAL learns both functional and structural indices to design analogues. The explanations in EBI are similar to those in EBL and EBG; the distinctions between these explanations and SBF models will be explained later. Explanations in EBI refer to the malfunctions of devices while SBF models are explanations of the functioning of devices.

In addition to the above differences, our model-based approach is also different from EBI in

the learning method itself although both use explanations for learning indices to cases. First, EBI assumes that a pre-enumerated set of indexing features is available, and its learning task is to select some subset of the set of features. In contrast, our approach assumes knowledge only about the *types* of features that are to be used as indices (e.g., knowledge that functions expressed in substance properties are used as indices) but identifies the exact indices from the SBF models. Second, EBI necessarily determines both irrelevant and relevant indexing features. In our approach, in contrast, one needs to determine only the relevant features. This is because the SBF model generated by its problem solver automatically rules out all irrelevant features. Third, our approach integrates model-based learning with similarity-based learning (SBL). Specifically, it uses the model-based approach to learn the indexing features and the SBL to generalize over the learned features.

10.3.2 Learning Abstractions from Experience

Many psychological studies in human learning show that people learn abstractions (generalizations) as they acquire experiences in a domain. For example, Wisniewski and Medin (1991) have found that people use domain theories in unsupervised learning of concepts from examples. In our approach, the case-specific models play a role analogous to domain theories in that they constrain the process of learning abstractions. In our theory, an agent predominantly performs unsupervised learning because it generalizes over its own problem-solving experiences. However, if the agent fails to create a successful solution to a design problem, then we provide the correct solution (and no other information) as feedback to the agent.

The proposal that learning from experience is facilitated by explanations of specific experiences dates at least as far back as Winston (1980). Winston's model assumed knowledge of "what" is the concept being learned and relied on information concerning whether an example is a positive instance or negative instance of the concept. Winston's later models (1982, 1986) show that learning can be done by analogically transferring causal links in the explanation of an example to the target concept. Winston's work addressed learning of a different type of abstractions than what IDEAL learns. His FOX and MACBETH systems learn rules from specific situations by climbing up a known classification hierarchy for objects and chunking over causal inference chains. The generalization process in this learning is constrained by making sure that the exercise (i.e., the new situation) is "close" to the rule to be learned. Similarly, in his early system ARCH which learns concepts from specific examples, Winston (1975) assumes that the difference between two successive examples is very small (i.e., difference in one feature). But, in IDEAL, we do not make such a simplifying assumption.

There are many AI models of learning from experience but most of them focus on the process of specializing some general knowledge to specific problems and making problem solving more efficient. Some other experience-based systems learn generalizations of indices but not of cases. Two exceptions are AQUA (Ram, 1993) and OCCAM (Pazzani, 1991) which make generaliza-

tions over specific cases. But few models have addressed learning of high-level abstractions of the *types* we described earlier. JULIANA (Shinn, 1989) and ASIS (Roverso et al., 1992) that we have already discussed learn different types of abstractions. We will discuss Birnbaum and Collins' work under "strategy learning" later as both our work and theirs deal with learning of abstract strategies.

ASIS discovery system (Roverso et al., 1992) also learns abstract models of specific situations. But these models are structural models and they are different from functional and causal models of generic mechanisms or processes. Further, ASIS's method retains the entire model of a specific situation for the abstract model and simply replaces each object and link by the corresponding abstract object and link. Moreover, these abstract models are not stored in memory for reuse; but in our approach, the abstract models are stored in memory for potential reuse in future.

Another model that addresses learning of abstractions from problem-solving experience is SOAR (Laird et al., 1987). However, the generalizations learned by SOAR are simple rules and are specific to types of subgoals in the problem.

10.3.3 Learning of Strategies

The problem of learning strategies has been studied by several researchers in AI since 1970s. For instance, (Waterman, 1970), (Mitchell et al., 1981), (Amarel, 1968), (Banerji and Ernst, 1972), (Ernst and Goldstein, 1982), (Mostow, 1982) and (Birnbaum and Collins, 1988). Keller (1982) analyzes some of these research efforts towards strategy acquisition in terms of the four important issues within the strategy learning problem. In his words, they are (1) learning strategic components from examples or problem-solving traces, (2) recognizing the applicability of a strategy, (3) customizing or instantiating an abstract strategy, and (4) transforming an existing strategy. From Keller's analysis, we can say that none of these research efforts have addressed all these four issues in strategy learning. In contrast, our work cuts across all these major issues, although not addressing so deeply into all these issues. IDEAL's task of learning GTMs from specific design experiences is an instance of the task of learning strategic components. Once an initial hypothesis for a new strategy has been formulated, the subsequent revision due to new design experiences is an instance of the strategy-transformation task. Of course, when IDEAL learns a new strategy, it needs to also acquire the applicability conditions for the strategy. And, finally, when it uses a strategy, since its strategies are abstract, it needs to instantiate the strategy in the context of a specific problem. Thus, IDEAL attempts to address all the above major issues in strategy learning. However, IDEAL's theory of how a new strategy is formulated from experience is the most interesting part to compare with earlier work.

Waterman (1970) used a self-modifying production system to address the issue of learning of heuristics for poker game-playing. His system has production rules that are composed of state vectors as conditions, and game decisions as actions. It can take input from two different sources: training information from a human or that generated by a machine from poker-playing

experience. IDEAL also handles input from multiple sources including these two. But, the content of input information is different in the two systems. The input to Waterman's program consists of specifying a game state vector, what is an acceptable game decision, what state variables are relevant to the decision, and why the decision is a good one in terms of constraints on the variables. Whereas IDEAL's input consists of specifying only a design that achieves a given function and how it achieves that function. Waterman's program uses the training information and modifies the existing rules (i.e., changes to the constraints on the variables in the condition part of the rules) or adds new rules so that they comply with the given data. Whereas IDEAL's output is a design pattern (in particular, GTM) that is abstracted over regularities in the given design and the retrieved similar design.

Mitchell et al. (1981) have followed up on Waterman's work and developed a system called LEX. Like Waterman's program, it also learns heuristics but for solving problems in a different domain, i.e., integral calculus. LEX knows a priori all the operators needed to solve calculus problems and also the conditions under which each operator can be applied. What it learns are control heuristics for operators that specify when they should be applied. In that sense, IDEAL is quite different because it does not know any thing about the strategies it learns; it not only learns the applicability conditions for a strategy but learn the strategy itself. There are two interesting aspects of LEX that are worth noting: (1) it generates its own training problems and analyzes its performance on them and (2) it represents the partially learned heuristics in version spaces (Mitchell, 1977). IDEAL differs from LEX on both these aspects—IDEAL cannot generate its own training problems and it does not represent versions of a strategy but rather it commits to one level of generalization. Such commitment in IDEAL is guided partly by the vocabulary of device models and partly by the similarities and differences between design experiences. Other important differences between LEX and IDEAL are that LEX uses meta-knowledge and problem-solving trace in its learning while IDEAL does not use either. Instead, IDEAL handles the difficult issue of determining what to learn by situating the learning task in problem-solving context (failure + feedback from oracle) and using the device models.

Of those models that have addressed learning of strategies, Birnbaum and Collins' (1988) work probably comes closest in sharing our goals—modeling the acquisition of abstract strategies that enable transfer of expertise from one domain to another. Their work concerns the use of EBL techniques in failure-driven learning of abstract strategies for planning in several domains by transfer of experience. GTMs in our work are similar to their abstract strategies such as "fork" in chess in that GTMs act as abstract plans for solving design-adaptation problems (Stroulia and Goel, 1992). Also, IDEAL's process of their use is similar to that of (Birnbaum and Collins, 1988). However, Birnbaum and Collins view the abstract strategies to be useful primarily in accessing a relevant experience, that is, they view cases to be indexed by these abstract concepts. Furthermore, IDEAL's method of learning the abstract strategies is completely different. Birnbaum and Collins' program uses problem-solving traces to reason about plan failures and to abstract strategies such as "fork." In contrast, IDEAL uses models and explanations

of how devices function to learn the abstractions. Also, we model the acquisition of other types of “abstract concepts” such as the models of GPPs in addition to those of GTMs.

10.3.4 A Model-Based Approach to Learning

As discussed earlier under “learning of abstractions from experience,” Winston’s model of learning (1982, 1986) is one of the earliest to advocate a “model-based” approach. In his theory, the causal relationships in an explanation of the experience are transferred to the target concept to be learned. Similar approach was used in Kedar-Cabelli’s (1988) work on purposive explanations in learning concepts. Her work however emphasized that the functional roles of elements in a plan are what need to be transferred in order to explain a new instance of a target concept and learn the target concept. The theory in IDEAL that concerns learning of models (by revision) is similar to these two. However, in IDEAL, both functional and causal relationships from the model of a known device are transferred and modified to learn the model of a new device.

Our model-based approach to abstraction is similar to explanation-based generalization (EBG) (Mitchell et al., 1986) and explanation-based learning (EBL) (DeJong and Mooney, 1986) in using explanations (SBF models) to constrain the learning of “concepts.” However, most EBL systems assume some knowledge of the target concept *a priori*; our model-based approach attempts to discover them. The learning task in IDEAL is further different from the tasks in EBG and EBL in that the latter address the task of learning from a single example while IDEAL learns from two examples, more similar to mEBG (multiple example EBG) (Cohen, 1988; Flann and Dietterich, 1989). The learning methods are also different in that EBG and EBL involve merely “chunking” over an explanation while IDEAL does abstraction over regularities in specific explanations. The distinction between these two processes is similar to the one drawn between *generalization* and *abstraction* by Michalski (1983).

Also, our model-based approach differs from EBG and EBL in the kind of explanations it uses. First, while the explanations in EBG and EBL are general such as a resolution proof in First-Order Logic (FOL) that can be of any type depending on context, the explanations in SBF models are of a specific kind, that is, the latter are functional and causal, and thus provide how causal processes result in the achievement of specific functions. Second, the explanations in EBG and EBL specify how an example is an instance of a target concept while SBF models are explanations of the functioning of devices. Third, the explanations in EBG and EBL are constructed at run-time from domain-specific rules whereas SBF models are formed by revising old models (or by a combination of model-revision and primitive-behavior composition) as part of the problem solving. Fourth, SBF models are grounded in a well-defined component-substance ontology.

Dietterich (1992) presents a model-based approach to learning models of plant physiology useful for the task of predicting how climate determines the location and distribution of plant ecosystems (biomes). The model of a plant is represented as an augmented finite state machine.

These representations are similar to ours to the extent that a behavior is represented as a chain or cycle of alternating states and transitions. Dietterich's approach assumes an initial model, although incomplete but general, and subsequently refines the models based on examples of plant ecosystems so that the refined models are more accurate for later predictions. In our approach, the direction of learning is from specific models (of given experiences) to generalized models of different kinds. We also give an account of how the specific model of a new device is formed by revising that of a source analogue. Also, Dietterich assumes that the examples are available in batch in contrast to our incremental approach.

10.3.5 Multistrategy Learning

Our work integrates the model-based approach with the similarity-based method for learning abstractions. Pazzani's OCCAM (1991) also integrates SBL, EBL, and theory-driven learning (TDL) for learning of concepts. OCCAM assumes knowledge of general patterns such as that of causality. It uses them for learning concepts, domain-specific causal relationships, and for acquiring explanations that help make a shift from relying on SBL to using EBL. The types of abstractions learned in our approach fall into the same class as these general causal patterns but are different; the major difference is that they are learned from experience in our approach. OCCAM uses general patterns of causality to acquire specific explanations; our approach uses the abstractions to form explanations (SBF models) of specific devices as it solves design problems by model-based analogy.

When multiple strategies are used for learning or reasoning, there is a need for strategy selection. IDEAL uses the model-based and similarity-based methods at one level and uses different, more-specific strategies at another level for its learning. At the top level, it does not make a selection between the model-based and similarity-based methods, instead, it uses these two methods for different subtasks of its learning of abstractions. But at more specific levels, it selects a strategy (such as abstraction over substance parameter transformation or substance location transformation for learning GPPs) dynamically depending on the type of functional transformation in the device models it compares.

In addition to OCCAM, these are a few other systems that do strategy selection for learning: AUTOGNOSTIC (Stroulia, 1994), Meta-AQUA (Ram and Cox, 1994), and Meta-TS (Ram et al., 1995). All these systems do meta-reasoning while IDEAL does not because its task is different and it does not need reasoning at that level. However, the strategy selection in those systems is relevant here. As we already discussed in earlier sections, AUTOGNOSTIC uses a common framework in several dimensions with IDEAL. AUTOGNOSTIC performs different learning tasks such as learning of new tasks in the problem solver it is associated with, learning indices to the domain knowledge of its problem solver, and learning new domain concepts for its problem solver. Its learning is driven by problem-solving failures and guided by the meta model, the model of its problem solver. It uses different strategies for different learning tasks. Given a

learning task, it chooses a learning strategy based on the conditions of knowledge availability in the failed episode. In that sense, it does a dynamic selection based on the specific conditions, similar to IDEAL.

Meta-AQUA is a reflective failure-driven learning system in the domain of story understanding. Its story understander uses cases and explanation patterns (XPs) to understand stories it reads. Some of the failures it recognizes are conflicts between the input and the expectations that are set up by its cases and XPs. It uses meta-level XPs (Introspective Meta-XPs), which capture prototypical error patterns, to find causes for the failures. Once it identifies the potential causes, it formulates a set of knowledge goals which specify "what" needs to be learned and "why" in the specific context. Then its task is to select an appropriate strategy from a set of learning strategies that can be used to achieve the knowledge goals. It uses a non-linear planner to accomplish this selection task. Thus Meta-AQUA's strategy selection is much different from IDEAL's. Meta-TS explores the same theory as that of Meta-AQUA in the context of trouble-shooting electronic circuits. It also uses the same kind of strategy selection as Meta-AQUA.

10.3.6 Multistrategy Reasoning

In this section, we compare our use of multiple strategies in reasoning and dynamic selection of strategies in IDEAL with other work. IDEAL uses multiple strategies for several different tasks in its computational process for analogy. For instance, it uses multiple strategies for design adaptation and selects them automatically at runtime. Its flexible control of strategy selection is motivated from and similar to the one used in another system called ROUTER, although the latter work is in a different task context. That is, ROUTER solves a route-planning task and not a design task (Goel et al., 1991; Goel and Callantine, 1992; Goel and Callantine, 1991).

ROUTER's task is to produce a path connecting two given locations in a geographical region. It describes a path in terms of a sequence of connected locations intermediate to the given initial and final locations. Each location is an intersection between two streets in a map. ROUTER uses two different methods of reasoning, namely, model-based reasoning and case-based reasoning. Its model-based reasoning method uses a hierarchical topographic model of the geographical region and does search within the parts of the region called neighborhoods. Its case-based reasoning uses cases of previously planned paths between locations. Given a path planning task, it dynamically selects between these two methods using some criteria for selection. In its application of either of these methods, ROUTER decomposes the given path-planning task into smaller path-planning tasks. For the recursive invocation of these subtasks again it uses the dynamic selection of the two methods. Thus, for a given task it may combine different sequences of applications of these two methods resulting in flexible control flow. IDEAL also uses a similar control of architecture for its selection of adaptation strategies for the given adaptation goal(s) in design. In IDEAL too, when subproblems are generated as part of adaptation, the subproblems

are solved by selecting appropriate adaptation strategies and combining the solutions of the subproblems.

10.3.7 Learning by Discovery

Discovery can be characterized as learning without knowing *a priori* what is being learned. In this sense, the learning tasks we are modeling can be viewed as discovery tasks because we do not assume *a priori* knowledge of the abstractions being formed from experiences. Since some abstractions can be physical principles and processes, our approach can be compared to work in scientific discovery such as BACON (Langley et al., 1987), FAHRENHEIT (Zytkow, 1987), and ABACUS (Falkenhainer and Michalski, 1986). These systems require a large amount of data because they use inductive approaches to discover regularities and form laws. In contrast, our approach involves incremental discovery of physical principles using models at lower levels of abstraction (e.g., device models) to guide the discovery process. Hence, it requires fewer examples for discovering useful principles. Most of the above systems use predesigned experiments to test their hypotheses.³ On the other hand, our approach takes a different stance on experimentation—it views *problem solving* using hypothesized “concepts” as testing the hypotheses. Thus experimentation is not planned but rather is a consequence of solving problems in the real world.

Learning of abstract models from several specific designs is also related to the tasks of theory formation and hypothesis formation—an abstract model is a theory of the functionality of the specific designs. In this context, our work can be compared to several systems such as PHINEAS (Falkenhainer, 1990), COAST (Rajamoney, 1988; 1990), and HYPGENE (Karp, 1989; 1990). PHINEAS is a discovery system that forms theories of time-varying physical behaviors by a single similarity-driven explanation mechanism, that is, by analogy to the explanation of a similar situation. The representations in PHINEAS are based on Forbus’ (1984) qualitative process (QP) theory. To the extent that the device-specific model of a new design is formed by revising the model of a similar experience (or by instantiating an abstract model in the experience), our work is similar in spirit to PHINEAS. PHINEAS uses the behavioral abstraction of a given situation to access a source analogue but assumes that the behavioral abstractions are known *a priori*. Our work differs from PHINEAS in modeling not only how abstract models can be used for accessing a source analogue but also how they can be learned by generalizing over experiences.

COAST models theory revision but not theory formation. COAST’s domain theories are also represented in QP and they capture processes in the domain of naive physics; hence they are of the same type of knowledge as device models in our work, more specifically, a device model corresponds to a component of theory in COAST. In our work, once a hypothesis (for the specific type of abstract model) based on experiences is formed, the subsequent revision

³Hypothesis formation and hypothesis testing (or experimentation) are two phases of discovery.

(generalization or refinement) of the hypothesis is similar to the task of COAST. COAST uses an explanation-based method for revising a theory when it encounters an anomalous observation. COAST's method of testing a proposed theory, namely, exemplar-based theory rejection, is very appropriate in our framework as well because in our approach prior experiences (or prototypes formed from them) play an important role in learning as well as other reasoning processes.

HYPGENE is a system that generates hypotheses in the discovery of attenuation, a mechanism of gene regulation in bacteria. HYPGENE views the task of hypothesis formation as a design task. HYPGENE uses qualitative representations for biological theories; the representations of models in our work also are primarily qualitative but can accommodate quantitative representations as well. HYPGENE generates a hypothesis by synthesizing from the primitives used to describe theories and the initial conditions of experiments. In contrast, our approach uses a model-based method to generate hypotheses by generalization over experiences.

10.4 Analogical Reasoning

Psychological data indicates that people often use analogies in solving problems of various kinds (Gick and Holyoak, 1980, 1983; Gentner, 1983; Ross, 1984). Computational models of analogy date as early as Kling (1971) though these early models were only approximate. Later computational models such as (Winston, 1982; 1986), (Gentner, 1983; Falkenhainer et al., 1989), (Kedar-Cabelli, 1988), (Carbonell, 1986), and (Holyoak and Thagard, 1989a) are more complete but still address only some stages of analogical reasoning, mostly the *mapping* and *transfer* stages. For instance, SME (Gentner, 1983; Falkenhainer et al., 1989), PDA (Kedar-Cabelli, 1988), and ACME (Holyoak and Thagard, 1989a) do not model analogue retrieval. Other theories such as ARCS (Thagard et al., 1990) focus only on the *retrieval* stage. In contrast, we consider all stages of analogy in this work.

Psychological studies also show that people transfer relationships between objects in the source domain and not their attributes (Gentner, 1983). Most theories of analogical reasoning also involve transferring relationships directly from a source analogue to the target problem: relationships that conform to systematicity principle as in (Gentner, 1983), or causal relationships as in (Winston, 1982; 1986), or functional relationships as in (Kedar-Cabelli, 1988) are transferred. Our work is similar to both (Winston, 1982; 1986) and (Kedar-Cabelli, 1988) in that IDEAL also transfers causal and functional relationships in a source domain to the target domain, although the tasks we model are quite different.

We organize our discussion of related work in analogy into four different parts: (1) purpose-directed analogy (covering Kedar-Cabelli's, Winston's, and Holyoak's work); (2) model-based analogy (covering Winston's, Falkenhainer's, and Roverso et al.'s work); (3) case-based approach (covering Hammond's, Koton's, Navinchandra's, and Shinn's work); and (4) psychological theories (covering Gentner's SMT, Holyoak's pragmatic schema model, and Ross's and Hintzman's exemplar models).

10.4.1 Purpose-Directed Analogy

IDEAL's analogical process that involves transfer between specific designs via specific SBF models is very much similar to the transfer process in Purpose-Directed Analogy developed by Kedar-Cabelli (1985, 1988), although the overall task in the latter is different. The task in her work is refining a known goal concept by explaining how a given instance achieves the purpose of the goal. The core idea in Purpose-Directed Analogy is to use the purpose for which the analogy was being made to determine what relations in a source analogue to transfer to the target problem. Similarly, in our model-based approach, the purpose of the target problem (i.e., the function of the desired device) plays an important role in analogical transfer: the transfer is based partly on the functional differences between the source and the target designs.

Purpose-Directed Analogy involves forming an explanation for the purpose of a source example using EBL and mapping the explanation over to the new example. One question is why can't the explanation for the new example be generated using the same EBL method to begin with?! Purposive explanation constrains what features in the example are relevant for revising the concept. Purpose-Directed Analogy assumes that the domain theory available is complete and correct, while IDEAL does not. In Purpose-Directed Analogy, the transfer is not only within-domain but also assumes that the purposes of the source and target are the same. Like Winston's work (1980, 1982, 1986) but unlike ours, purpose-directed analogy also covers only simple modifications to source explanations and assumes much prior knowledge of the target problem and domain. Purpose-Directed Analogy does not address the issues of retrieval and storage of analogues. In contrast, our model-based approach addresses several stages of analogical reasoning and makes few assumptions about the target problem. The latter difference is due in part to the design task we are modeling—design generation is much more open-ended than explanation completion and allows for fewer assumptions about prior knowledge.

IDEAL also shares the most with another cognitive model, that is the pragmatic schema model (Gick and Holyoak, 1983; Catrambone and Holyoak, 1989). In this model, the pragmatic structure of the source analogue (i.e., sub-goals that contribute to the solution of the problem) guides the transfer. An analogous situation for IDEAL is that the functional structure and decomposition of an SBF model of a device guide the transfer process in IDEAL. Their work involves strategic learning of the schemas, which are different from IDEAL's abstractions. IDEAL shares the ideas that the learning occurs from comparison of two analogues, and that the schema induction is important for cross-domain transfer. Some of their experiments show that explicit hints facilitate better schema induction and higher rates of transfer. However, in IDEAL such hints are not given. Holyoak and Thagard's (1989) PI system is a computational model that incorporates some of these findings and covers schema induction and learning of general rules.

10.4.2 Model-Based Analogy

Winston's FOX and MACBETH: Our theory is similar to Winston's (1980, 1982, 1986) cause-directed analogy in that both models emphasize that the causal relationships in a source analogue determine and constrain what is transferred from the source to a target problem/situation. However, Winston's work focuses on the task of explanation completion and query answering that are much less open-ended than design generation task we address in IDEAL. It is because in the prior task, there is a lot more information directly available in the input to the task than that in the design generation task. Whereas the output information to be inferred in the explanation completion and query answering task is much less compared to that in the design generation task. Although IDEAL shares the motivation that the knowledge to be transferred from a source to a target is the causal relationships in the source analogue, it does focus, in addition, on the functional relationships between the source and target.

The transfer task in Winston's systems (FOX and MACBETH) is to identify the mappings between the individual items in the source and target situations by a process of exhaustive matching and transfer the causal relationships between the matched individuals in the source to the corresponding ones in the target. By that process, his systems infer values for unknown variables (which could be physical variables such as voltage or volitional such as an agent depending on the domain), typically only one in an instance. In contrast, IDEAL's task does not involve identification of mappings explicitly because that occurs as a by-product of retrieval itself. And, it is because of the uniform representations of functions in SBF ontology in all the analogues and because of using a structured matching on those functions for retrieval. Unlike in Winston's systems, IDEAL's task involves inferring values for multiple unknowns, as it does design generation. The transfer task in IDEAL however involves copying the functional and causal relationships in the model of the source analogue over to the target problem, and then modifying them according to the differences in the functions. While Winston's systems too can handle modifications to source analogues, they are much simpler than those that IDEAL can handle. For instance, his work addressed only how simple modifications (such as substitution of relations and class specialization in a source analogue by corresponding ones from target) can be done to a source analogue in order for it to fit the target situation. In his work, the distance between a source and a target is typically very small. In contrast, IDEAL addresses the issue of complex, non-local modifications that involve patterned insertions of new design elements into the source analogue. Recall from the Chapter 5 that IDEAL performs two different kinds of transfer: (1) transfer from a specific source situation to a specific target situation and (2) transfer from an abstraction to a specific target situation. Although Winston's work primarily looks at the first kind of transfer, that is, transfer between specific situations, his theory accommodates the possibility of general rules themselves being transferred as source analogues. While IDEAL's theory attempts to address the issue of cross-domain retrieval, Winston's theory does not. We have discussed Winston's work earlier with respect to the relationship between

the learning tasks he addressed and that IDEAL covers.

In contrast to (Winston, 1982; 1986) and (Kedar-Cabelli, 1988), our theory of model-based analogy (MBA)—access an abstract model in the source domain and instantiate it in the target domain—is also based on transferring functional and causal relationships via shared mental models at higher levels of abstraction. In this respect, our theory is more similar to NLAG (Greiner, 1988), PHINEAS (Falkenhainer, 1990), JULIANA (Shinn, 1989) and ASIS (Roverso et al., 1992). However, both NLAG and PHINEAS assume that the abstractions are known *a priori* and do not address the issue of acquiring them. NLAG views a source analogue as an instance of one of its domain-independent abstract models and instantiates the relevant abstraction in the target domain.

PHINEAS: PHINEAS is an analogy-based machine-discovery system that learns qualitative explanations for time-varying physical behaviors. PHINEAS uses the behavioral abstraction of a target situation to retrieve a similar abstraction and the specific experiences stored under that abstraction. It selects one of the experiences based on further matching to the target situation and maps the selected one. Therefore, our work is similar in some respects but interestingly distinct in some others. For instance, while PHINEAS thus uses the known abstractions for accessing a relevant specific experience and its explanation, IDEAL uses the abstractions themselves for transfer. Recall that IDEAL's process involves two kinds of transfer: one between the specific designs and the other from an abstraction to a specific design situation. The first of these two transfers is similar to PHINEAS's task. And, the second is not addressed in PHINEAS at all. Although the first kind of transfer task is similar, PHINEAS uses a very different kind of method, which is a syntactic, content-independent approach that relies on systematicity principle (Gentner, 1983; Falkenhainer et al., 1989). In this approach, PHINEAS constructs all possible mappings between the source situation and the target situation, while IDEAL establishes the appropriate correspondences not by explicit mapping but rather by structured matching on uniform representations during retrieval itself. Thus the correspondence problem is not a major issue in IDEAL while it is so in PHINEAS. Also, while PHINEAS explores issues in analogy in the context of the task of explanation completion which is a much less open-ended task than design generation that IDEAL uses. Finally, PHINEAS does not address the issue of learning of abstractions.

ASIS: Model-Driven Analogy, as implemented in ASIS, is another approach that also accounts for the acquisition of abstract models. ASIS forms abstractions from target descriptions while accessing a source analogue. Our approach, on the other hand, covers both learning of abstractions while storing experiences in source domain and during solving problems in a target domain. The rationale is that it seems implausible that an agent always can form abstractions *before* solving problems in a domain, especially for the complex problems such as design tasks. It appears more plausible that an agent may have ability to recognize the applicability of an already acquired abstraction to the current problem. Further, it appears more plausible that an agent can learn abstractions from experiences in a familiar domain, and acquire experiences in

a new domain by analogical transfer.

10.4.3 Case-Based Approach

IDEAL covers both within-domain analogies and cross-domain analogies.⁴ IDEAL performs two different kinds of transfer: (1) transfer from a specific design to a specific problem and (2) transfer from an abstraction (i.e., a design pattern such as cascading GTM) to a specific problem. The first type of transfer task is equivalent to the task of case adaptation in case-based reasoning (Kolodner, 1993). For that reason, IDEAL views the transfer task as the modification of the source analogue to meet the specifications of the target problem. Several researchers (Ashley and Rissland, 1988; Hammond, 1989; Kolodner and Simpson, 1989) have developed computational models for case-based reasoning that posit different methods for adapting previous cases for solving new problems. For example, *heuristic search* (Stallman and Sussman, 1977) and *heuristic association* (Hammond, 1989). The case-based method itself has been recursively used to adapt cases (Kolodner and Simpson, 1989).

In addition, several researchers have used model-based methods for case adaptation. For example, Koton (1988) has used causal domain models for comprehending diagnostic problems in internal medicine and retrieving appropriate diagnostic cases from memory; Simmons and Davis (1987) have used causal domain models for debugging plans but only for testing modifications to a plan, not for generating the modifications; and Sycara and Navinchandra (1989) have proposed the use of causal domain models for elaborating engineering design problems, retrieving appropriate cases from memory, and adapting them. Unlike them, we advocate the model-based approach for several subtasks in analogical reasoning: retrieval of similar analogues from memory, generation of modifications to the retrieved analogue, execution and evaluation of the generated modifications, and finally, storage of new analogues in memory. Furthermore, IDEAL's SBF models are different from the causal models of Simmons and Davis, Koton, and Sycara and Navinchandra. The behavioral states and the state transitions in their models are grounded neither in the function nor in the structure of the system. In contrast, the SBF model explicitly relates the internal causal behaviors to both the function and the structure of a device, and thus constrains them both from the top and the bottom.

The second type of transfer in our theory is similar to Shinn's (1989) work on abstractional analogy in several respects. That is, our work shares the basic assumption of JULIANA that an analogy between the source and the target can be performed through a common abstraction, especially when the source and the target domains are far apart. JULIANA emphasizes that common abstractions, if they do not already exist in memory, are learned during the mapping process while our work accommodates learning during transfer as well as storage. However, the current version of IDEAL implements and tests only learning during storage. Although Anderson's psychological data (1986) denies a separate learning process after analogical problem

⁴Of course, it inherits its basic ability to do within-domain analogies from its predecessor systems KRITIK/KRITIK2.

solving, we propose that discovery of the kind we are modeling might occur as an "after-the-fact" process (or, in retrospect), more specifically, while storing an experience in memory.

Further, JULIANA assumes that given a target problem, a source case, however partially it might match, can be retrieved. This seems plausible only in two situations: either there is already a common abstraction (in terms of indices) engineered into its memory or the distance between the target problem and the source case is very "small." In contrast, our approach takes a different view in which initial abstractions are hypothesized from experiences that might be close to one another or that are from the same domain. These abstractions may be retrieved and instantiated in different target domains to make cross-domain analogies. Furthermore, they may be subsequently revised based on feedback when they are used in later situations.

Within the case-based approaches, although Hammond's (1989) work concerns with a different task and a different adaptation method compared to our work, some of the subtasks of CHEF relate to those in IDEAL. In particular, IDEAL's tasks of understanding failures coming from an external evaluator and reformulating problems in order to redesign are related to CHEF's tasks of understanding plan failures and replanning. There are, of course, some similarities and some differences between the specific tasks, methods, and knowledge assumptions in IDEAL and the corresponding ones in CHEF. First, while CHEF has a built-in plan simulator that is equivalent of real-world execution, IDEAL does not have any such. But, IDEAL does have an internal evaluator that uses its knowledge of the device model to "mentally" simulate a candidate design, which is different from CHEF's real-world simulator. IDEAL however relies on an external agent to provide feedback on the outcomes of executing the candidate design in the real world. IDEAL's policy is more valid than CHEF's because one could criticize CHEF rightfully as follows: if CHEF's simulator already has the necessary knowledge to verify the plan, why can't it use the same to produce a correct plan in the first place and why does it have to deal with failures at all?

Second, both IDEAL and CHEF share the same argument that it is not enough to identify or know a failure in a candidate design/plan, but rather there is a need for explaining the failure. While in CHEF, a failure is described only as an undesired final state, the design failure in IDEAL can be either an undesired state or an undesired state transition. IDEAL and CHEF use different methods for forming explanations of failures. IDEAL uses its knowledge of generic models (in particular, GPPs) to form causal explanations of the failures in the context of the candidate design. In contrast, CHEF uses a set of inference rules that relate plan steps and domain objects to chain through the plan steps and states and forms a causal explanation (much like in the way explanations are formed by goal regression in EBL/EBG).

Third, both IDEAL and CHEF fix the failures, but each differently. CHEF uses the causes of the failure to describe the planning problem in terms of general causal vocabulary and uses that description to access plan-debugging strategies. CHEF uses TOPs (Thematic Organization Packets) to encapsulate mappings between planning problems and applicable set of repair strategies. In contrast, IDEAL uses the causes of the failure to reformulate the problem by discovering

any new constraints that are still in the specific vocabulary of the problem. It is because IDEAL uses those new constraints to access designs for composing with the failed design. Although IDEAL too has knowledge similar to TOPs in its GTMs, it does not use them to repair the failed designs, instead, it uses them for adaptation.

Fourth, while CHEF learns from its failures to anticipate them in future and avoid, IDEAL does not do so. CHEF is able to learn because it assumes that the failures are such that it can find features in the initial problem that are predictive of the failure. But in the kinds of failures IDEAL deals with, such a "blame assignment" is not possible because the failures may be due to interactions of the device with new environments and the initial problem may not specify anything about the environment. But since IDEAL has reformulated the problem, it can store the new design indexed by the reformulated problem so it would be used only in the right situations. In contrast, CHEF indexes its new plan by the failures as well as goals, and thus restrict the use of the plan for the right situation.

10.4.4 Psychological Theories

In this section, we compare IDEAL with the three major psychological theories of analogical transfer, which are Gentner's (1983, 1989) Structure-Mapping model, Holyoak (1984,1985) and Holyoak and Thagard's (1989) pragmatic schema model, and Ross's (1987) and Hintzman's (1986) exemplar models (Ross's reminders theory and Hintzman's multi-trace model). We compare on the three issues in analogical transfer that are considered important in psychological theories (Reeves and Weisberg, 1994). In addition to the basic papers on the three models, we draw on Reeves and Weisberg's (1994) extensive survey of the psychological theories of analogy.

Before considering the three issues, let us list the three types of knowledge that may have different roles in analogical transfer:

1. **Problem Content:** The problem content consists of the semantic domain and the surface elements of a problem. The semantic domain is defined as the superordinate classification of a problem topic. But the inclusion of semantic domain in problem content is debatable. In IDEAL, the surface elements are specific substances and components, and the semantic domain is the functional classification of the design problems.
2. **Abstract Information or Structural Details:** This consists of solution principles (that can be stated as either formulas or propositions in formal domains; or schemas (Gick and Holyoak, 1983) or deep structure (Gentner, 1983, 1989) in nonformal domains). Schemas are also considered to have information about how to classify problems. In IDEAL, the abstractions can be design patterns such as generic physical processes or generic teleological mechanisms. The classificatory information in IDEAL's abstractions are their indices.
3. **Experimental Context:** This includes the setting, the experimenter, and the tasks in the experiment (in a psychological study). Since work on IDEAL does not involve any psy-

On the other hand, when IDEAL learns GPPs, it does store the new analogues because these learned abstractions are of a different kind, used for a different task, and when applied to the source analogue do not produce the needed solutions. But instead, they themselves can be used as “analogues” and can be instantiated in a target problem.

Issue 3. What is the relative importance of surface and structural-schematic elements of base and target problems in retrieval and application processes?

Most of the models under discussion suggest that both surface and structural-schematic features play a role in retrieval except the reminders theory which says that only surface features are important. We should note however that Gentner, Ratterman and Forbus (1993) tease out the relative importance of the two types of features in the subtasks of retrieval, namely, selection of all matching analogues and ordering them to choose the best. They indicate that surface features determine the selection of analogues but the structural-schematic features determine the ordering of analogues. In IDEAL too, both types of features are considered important.

In IDEAL, the indices to design analogues are provided by the vocabulary of SBF models which are “deep” (or semantic). But the problems themselves (i.e., the desired functional specifications), although specified in this vocabulary, contain surface elements (which we refer to also as domain-specific features) such as substances and their properties, and components and their properties. Thus, these surface elements determine the retrieval of source analogues. However, an alternative indexing scheme in IDEAL is based on “deep” features such as primitive functions (e.g., ALLOW, PUMP, CREATE etc.), which are part of the same vocabulary. Although this alternate indexing enables retrieval of analogues from a different domain, it imposes the requirement of eliciting “deep” features (i.e., these primitive functions) from the given problem specification that specifies only “surface” elements before retrieval can be done.

With respect to the question of which features are important in application of a retrieved analogue to the target problem, majority of the models agree that the structural (deep) features are more important than the surface features. The exception, of course, is the exemplar model—while multi-trace model is silent about this issue, reminders theory suggests that it is largely surface features that play a role in application.

Although IDEAL agrees with structure-mapping and pragmatic schema models on the second part of the question, what IDEAL considers as structural features is different from that in the other two models. In IDEAL, the knowledge of device models (that relate function, structure and behavior) is considered deep knowledge. The application of source analogue to target problems in IDEAL is guided by the “deep” knowledge of device models and the functional differences between the source and target problems.⁶

Table 10.1 reproduces the summary of the comparison of the three major psychological models from (Reeves and Weisberg, 1994). But it also shows IDEAL’s position on the particular issues in comparison to the other models.

⁶But, of course, the surface elements in the two problems also matter in the specific modifications done to the source analogue.

Table 10.1: How each of the above models of analogical transfer (including IDEAL which is a computational model) answers the critical theoretical questions

Theoretical issue	Structure mapping	Pragmatic schema	Multiple trace	Reminders theory	IDEAL
Automatic vs. Strategic	Automatic discernment of structure (not really abstraction; no schema induction)	Strategic	Both	Strategic	Both
conservative vs. eliminative	probably conservative	Conservative	Cons.	Cons.	Both (different in different contexts)
Does conservative induction include preserving episodic or contextual cues?	No	Posthoc explanation	Yes	Yes	Yes (only in the form of a partial spec. of the problem-solving task.)
Surface vs. structural features more important in:					
Retrieval?	Both (Surface in selection subtask and structural in ordering)	Both	Both	Surface	Both
Application?	Structural	Pragmatic structure	—	Largely surface	Models (semantic structure)

chological studies, the equivalent of this in IDEAL would include the "state" of IDEAL's memory at any given time and the particular problem-solving task (for instance, a characterization of the adaptive design task in terms of functional differences between a known design and a desired design).

The three major issues in analogical transfer are (1) whether schema induction occurs in analogical problem solvers, and if so, whether it is automatic or strategic? (2) whether any induction from examples (or problem-solving experiences) that occurs is conservative (with respect to retaining surface details and episodic details) or eliminative (i.e., those details are discarded)? and (3) what is the relative importance of surface and structural-schematic elements of source and target problems in retrieval and application (i.e., transfer) processes?

Issue 1. Many theories agree that schema induction from exemplars can occur. But the issue is whether it is automatic or strategic? In automatic induction, the cognitive system automatically tabulates the degree of overlap among several similar problems presented, and stores the composite of overlapping features as a separate problem representation. While in strategic induction, the schema induction is based on (a) explicit comparison of two or more analogues for likeness, (b) active processing of the schematic principle of one or several exemplars, or (c) the use of one problem to solve another.

In structure-mapping model, there is really no abstraction or schema induction, but rather the process involves extracting structural details from specific examples. The structure-mapping theory considers that process to be automatic. Whereas, the pragmatic schema model does consider schema induction that involves abstracting over the pragmatic goal structure in achieving the solution to a specific problem. This model considers schema induction as a strategic process that occurs during problem solving. Although the exemplar models do not give importance to schema induction, they nevertheless admit the possibility that abstractions may be formed after a large number of exemplars are seen. Multi-trace model suggests that schema induction can be both automatic and strategic, but the reminders theory suggests that it can be only strategic. In contrast to these theories, we suggest that the induction can be automatic or strategic in different, particular conditions. IDEAL's theory enables us to predict clearly what those conditions are that perhaps can be tested for psychological validity.

In IDEAL, an analogous issue is whether learning of abstractions occurs at storage time (which is automatic) or at problem-solving time (which is strategic). But, it is not clear whether learning at storage time is really automatic in IDEAL, because it involves an explicit comparison of two analogues for similarities and differences, which can be triggered by both "failure" at problem solving or "success" at it! Or, perhaps, learning triggered by failure is strategic, and that triggered by success is automatic?

Hence, one prediction IDEAL makes is as follows. Failure at problem solving may be the trigger for explicit comparison of analogues which leads to strategic abstraction, even if it occurs at storage time; if it occurs at problem-solving time, it is strategic anyway (following the Holyoak

et al's pragmatic schema model). Furthermore, IDEAL proposes that abstraction from analogues after successful problem solving is automatic because there is no "strategic" point to that learning (i.e., by this abstraction, it is not trying to avoid a failure in future problem solving nor is it trying to get better at problem solving later on).

Issue 2. Is the induction conservative (i.e., the surface details of exemplars and the episodic details of the learning situation are maintained after schema abstraction) or eliminative (i.e., those details are discarded)?

All the three major models agree on their answer to this issue and all of them suggest that it is conservative. But they differ on the related question as to whether conservative induction include preserving episodic or contextual cues. While exemplar models indicate that the contextual cues are preserved in induction, structure-mapping model suggests that they are not. Pragmatic schema model admits on a posthoc basis that the contextual information is perhaps maintained in the induction. In contrast to all these models, IDEAL's theory suggests that the induction can be both conservative and eliminative in different contexts.

There is one thing not clear in this conservative vs. eliminative induction. That is, it is not clear from these definitions whether the exemplars themselves are discarded in eliminative induction, or it means that the surface details are "not included" in the abstracted schema but the exemplars may be available as independent representations? In IDEAL, when learning at failure, it has been eliminative induction, and is in the sense that the specifics of analogues are discarded from the abstraction and also that the new analogue may or may not be stored.⁵ But, when learning from successful problem solving, it has been conservative induction, and is in the sense that the specifics are maintained separately from the abstractions. That is, both specific analogues and the abstractions are available, but they are accessed for different subtasks in IDEAL: analogues when a design problem is given, and abstractions when a retrieved design is being adapted.

The rationale behind the decisions in the current implementation of IDEAL is as follows. IDEAL learns GTMs when failed (i.e., either it was not able to produce a solution or it took more than one step of adaptation). It does not need the new analogue later on because it can apply the learned GTM to the source analogue and produce the needed solutions. One question that might arise is how will IDEAL know whether to discard the exemplar or not. One way by which IDEAL can decide this is by trying to use the abstraction immediately to solve the current problem and checking if it can generate the current exemplar—if *yes*, it can store only the abstraction, else it can store both the abstraction and the exemplar. This, we believe, follows the principle of cognitive economy. But, we are also finding that if learning were to occur at the problem-solving time rather than at the storage time, it is necessary to store the new analogue (with different indices as well as the functions)—because that would need to be accessed in later problem solving at the time of which the abstractions may be learned.

⁵In the current implementation, the new analogue is not stored, but in general, there is no commitment to not storing it.

IDEAL seems to share more ideas with the Pragmatic schema model than the others. But it differs from this too on some of the major issues in analogical transfer as indicated in the table. In all the three major models, the application involves transferring solution from a "specific" exemplar to the target problem (i.e., a direct transfer), although different abstractions help in different ways in these models. But, IDEAL similar to the pragmatic schema model acknowledges the possibility that the application may be directly from an abstraction to the target problem (by instantiation), which is important in cross-domain transfer. None of the empirical evidence on which the major psychological theories are based really "rule out" this possibility; it is perhaps a worthy psychological prediction to test! Nevertheless, our theory also accommodates the other possibility where the abstractions act as indices to specific exemplars and the application is from specific to specific (via learning the abstraction at problem-solving time rather than the direct transfer).⁷ The continuum between these two ends, i.e., transferring from specific to specific vs. transferring by instantiation of a shared abstraction, will determine the kind of transfer possible in the continuum from within-domain to cross-domain analogies.

On Empirical Evidence:

There is quite a bit of empirical evidence for the theoretical issues listed in Table 10.1. It is no surprise that there was evidence for contrasting positions, but it is surprising that the studies cited, even put together, have not ruled out some possibilities in arriving at their conclusions; these other possibilities seem to fall out of IDEAL so naturally! It was also surprising to me to see that most evidence was for the support of reminders theory, which we don't see having a potential to explain cross-domain transfer, because in this theory mostly surface features matter in various stages of analogical transfer!⁸

Regarding the issue of whether the schema abstraction can occur from a single example or whether it requires more examples, most researchers believe for the latter. However, there is some evidence for the former to occur under specific conditions, i.e., abstraction from a single example can occur when an explanation of the solution is available (e.g., Elio and Anderson, 1981) or when it is being mapped onto a target problem. We would claim, however, that in some tasks like design, it may not be possible to abstract from one example (i.e., a single design) even though an explanation is available. (Note that IDEAL learns from two or more design examples.) This is true especially for the kinds of abstractions such as generic mechanisms that capture regularities among examples. In such tasks, it is also not possible to map the source analogue onto a target problem across domains without having the abstraction. Another issue is whether the abstraction is automatic or strategic. Most evidence appears to be in favor of strategic abstraction, but automatic abstraction is not completely ruled out; automatic abstraction may be possible with a large enough number of base analogues.

Many experiments (e.g., Catrambone and Holyoak, 1989) have found that providing a schematic statement (i.e., a priori knowledge of what schemas may be learned) or diagram

⁷This part of the theory is only analyzed, but not implemented yet.

⁸This has been confirmed in a conversation with Brian Ross.

along with several problem exemplars leads to both better schema induction and higher rates of transfer than supplying source analogues alone. But an alternative way of facilitating better schema induction from source analogues alone (as done in IDEAL) might be to provide the analogues in the context of solving problems within a domain, in particular, when problem-solving failures occur. Hence, no explicit schematic statement may be needed to enable better schema induction.

Many experiments show that surface details influence retrieval and application of source analogue to target problems, even after some schema induction, which shows that they are retained. Some experiments (e.g., Keane, 1987) show that similarity of surface elements and semantic domain influence retrieval of source analogues in an additive manner. It was also found that surface elements did not need to be identical, but rather only similar, to facilitate retrieval. Most researchers agree that lack of surface similarity between source and target problems in cross-domain transfer which deters retrieval of a source analogue is one reason why cross-domain analogies are rare—this is considered a good indication of maintaining surface details. But, in those cases of successful cross-domain transfer, the source analogue may not be retrieved, but instead the abstraction may be retrieved and instantiated in the target problem, as in IDEAL! Then it is possible that the surface details need not be maintained at all or that the source analogue may be remembered through the abstraction. Also, that the transfer may not occur directly from the source analogue but instead might occur from the abstraction.

Some experiments show that contextual factors also affect the retrieval and transfer of a source analogue, i.e., the similarity in the contextual features between the source and target problems determines the retrieval. In design (or perhaps any task), the subtask in which the source analogue was acquired can be a “contextual cue” as well as the conditions of failure at that time can be. In IDEAL, if learning of abstractions were to occur at the time of problem solving, it appears that the similarity in contextual features (i.e., a partial characterization of the problem-solving task in terms of functional differences) are important for the retrieval of the source analogue. The three different types of cues for the retrieval of source analogues are: surface cues, structural cues, and contextual cues.

While many studies indicate that both surface and structural details affect retrieval of source analogues, they affect different substages of retrieval—surface details in “accessing” (i.e., selecting in our terminology) source analogues and structural details in “selecting” (i.e., ordering to choose the best, in our terminology) a source analogue for mapping. It also appears that their effect may be different in different domains—Bassok and Holyoak (1989) found that the rate of transfer from algebra domain to physics was much more than the transfer in the other direction. The explanation given for those observations is that the students realize algebra formulas are meant to be abstract and applicable to several different topics and so they represent them accordingly. Anyhow, the point we would like to draw from here in support of IDEAL is that it is the abstractions that are important to facilitate cross-domain transfer.

There is contrasting evidence for the role of surface details vs. structural details in the

mapping stage—Gentner et al. (1993) found that structural details matter most while Ross (1989) found that surface details matter most. However, there seems to be a shift in using the surface details to using the structural details as one gains expertise. In IDEAL, the issue of shift from novice to expert is not addressed. But, both surface and structural details matter in the application of a source analogue to a target problem in IDEAL.

CHAPTER XI

CONCLUSIONS

We revisit each of the major issues this research poses and addresses, and summarize its contributions to those specific issues. We begin by reviewing the overall problem we addressed. The research problem is to investigate the following dimensions of innovation in design:

1. non-local modifications to previous designs, i.e., changes to the topology of the device structure
2. cross-domain transfer, i.e., transfer of design knowledge between distant domains such as electric circuits and mechanical controllers
3. reformulation of design problems, i.e., the revision of problem specifications during the design process by modifying initial constraints and adding new ones

This characterization of innovative design raised the following issues:

1. **Content and Representation of Design Knowledge:** What design knowledge might enable these three aspects of innovation? That is, what might be the content of design knowledge and how might it be represented?
2. **Access and Organization of Design Knowledge:** How might such design knowledge (as in (1) above) be accessed/retrieved from memory when a new problem is given? What kind of organization and indexing might support better retrieval of relevant knowledge from memory? How might the design knowledge be related to other types of knowledge, if any?
3. **Use of Design Knowledge:** How might the design knowledge (as in (1)) be used for innovative design? That is, what might be the specific processes by which that knowledge enables the three aspects of innovative design?
4. **Origin and Acquisition of Design Knowledge:** Where does such design knowledge (as in (1)) that enables innovative design come from? How might it be acquired?

We addressed these issues by developing a theory of design patterns, a useful class of design abstractions, and representing design patterns using SBF models. In addition, taking a memory-based view and model-based approach also led us to address the issues of the acquisition of device

models, indexing and organization of design knowledge (in particular, design analogues, device models, and design patterns) in memory, and retrieval of those types of design knowledge.

11.1 Issues Addressed

In this research, we have addressed the following eight major issues within the overall goal of developing a theory of innovative adaptive device design. Put together, these issues cover the different stages in the computational process we proposed for design by analogy, i.e., the MBA process (see Figure 2.1).

1. What might be some useful high-level abstractions in device design that enable non-local modifications, cross-domain transfer, and problem reformulation? What might be the content of those abstractions and how might it be represented?
2. How can the useful abstractions such as design patterns enable non-local modifications to known designs in solving adaptive design problems?
3. How can those design patterns enable cross-domain analogies in design?
4. How might those design patterns enable understanding of external feedback on the evaluation of designs and enable problem reformulation based on that?
5. How can those design patterns be acquired automatically?
6. How might the SBF models of new designs be acquired automatically?
7. How might a new design analogue and a design pattern be indexed and organized in memory for later use? How can their indices be acquired dynamically?
8. How can design analogues and design patterns be retrieved from memory? What features in design problems determine the retrieval of design analogues and design patterns?

11.2 Contributions

We now present the contributions of this research to the specific issues addressed.

1. **Content Theory of Design Patterns:** The first question in building a theory of innovative design was what might be some useful abstractions in device design that enable the three aspects by which we characterized design innovation. In response to this question, we developed a theory of a particularly useful class of abstractions that we call *design patterns*. Then the question became what might be the content of those design patterns and how might it be represented?

Design patterns capture design knowledge in abstract, generic or case-independent terms. Depending on the specific type of knowledge, such as spatial, temporal, functional, and causal relationships among design elements, there can be different types of design patterns. Within the focus of this research (i.e., design of physical devices), we identified two different types of functional- and causal-type design patterns. Specifically, they are generic teleological mechanisms (GTMs) and generic physical processes (GPPs).

Furthermore, in this research, we provided a content theory of these design patterns and showed that the SBF language (Goel, 1989) is sufficient to represent these patterns as generic models. A GPP is a causal-type design pattern which captures patterns of relations between the output and the internal behaviors of physical devices. An example of a GPP is the process of heat flow. A GTM is both a functional- and causal-type design pattern which captures patterns of relations between the functions (a subset of output behaviors) and the internal behaviors of devices. Examples of GTMs are cascading and feedback mechanisms. Since design patterns capture only patterned relationships between design elements that do not refer to any specific device's physical structure, the function and behavior aspects of the SBF language were especially useful for representing GPPs and GTMs. Finally, this research provided a process account of how these two types of design patterns enable innovative design—a process that involves retrieval and instantiation of design patterns in the context of new problems.

2. **Non-Local Modifications:** Our characterization of innovative adaptive design includes the ability to make non-local modifications to known designs as one of the defining elements of our theory. There were three issues with respect to a theory of non-local modifications: (1) what are non-local modifications? (2) why are they necessary and important? and (3) how can they be enabled in device design?

First, non-local modifications are changes in the structural topology of devices. In contrast, the device topology remains the same before and after a local modification. The topology of a design refers to the arrangement of the design elements, that is, the configuration of the connections among the elements. Changes to a design's topology include addition or deletion of design elements, and connecting them in a different way.

Second, since most current theories of adaptive design address only local modifications (i.e., parametric tweaks) to known designs, the need for non-local modifications may be questionable. This research shows that they are necessary and important for two reasons and under specific conditions: past designs with the needed structural topology may not be available and thus simple, local modifications may not suffice; and even when a design with the appropriate topology is available, specific design elements with the desired functions may not be available to replace elements in the known design.

Third, the knowledge of design patterns can enable non-local modifications to known designs in solving adaptive design problems. This research, in particular, demonstrates the use of GTMs for the purpose of making non-local modifications in device design. We described a computational process in which the GTMs can be retrieved and instantiated in the target problems to solve different classes of design problems that require non-local, structural changes to the device topology in known designs. More specifically, the MBA process involves instantiating GTMs in the content of SBF models of known designs and modifying the designs to the target problems. That is, GTMs are used to solve the subtask of achieving adaptation goals in the MBA for device design.

3. **Cross-Domain Transfer:** The second dimension in our characterization of innovation in design is the ability to perform cross-domain transfer of design knowledge. Similar to the three issues raised with respect to a theory of non-local modifications, there were three issues with respect to cross-domain transfer as well: (1) what is cross-domain transfer? (2) why is it necessary and important? and (3) how can it be enabled in device design?

First, the notion of cross-domain transfer is dependent on what we mean by a "domain." The notion of a domain is fuzzy and there is no good characterization of domain in the analogy literature. In this research, we attempted to provide a working definition for domain. We characterize a design domain in terms of the structural elements available in it, such as pipes and pumps, and wires and batteries, at a given level of abstraction. Due to the fuzzy notion of domains, there is a continuum between what is within-domain transfer and what is cross-domain transfer. Cross-domain transfer in design involves the transfer of design knowledge obtained from experience in solving problems in a source domain to solve problems in a different domain.

Second, in this research we argued that cross-domain transfer is needed because knowledge of past designs generally may not be available in the same domain as a given target problem. Often it may be necessary to solve new problems by transferring expertise from a different domain. Although there was some previous work on analogical reasoning, much of it focused only on within-domain transfer or only on cross-domain transfer. Even that past work which focused on cross-domain transfer explored only some methods while some others are possible and necessary. For instance, the transfer of a design in one domain directly to solve a problem in a different domain (i.e., a domain that refers to different structural elements than those in the available design) is not possible or is difficult when the elements are distant in the known classification hierarchy (because the matching of elements depends on finding a common class for the elements). An alternative method that was also explored to some extent in the past involves transfer mediated by high-level abstractions shared between the source and target domains. However, in this method, although the abstractions play an important role, a specific design is nevertheless accessed in a source domain and the transfer still occurs from the specific design to the target problem.

Furthermore, in the previous work, there was no account of where the abstractions come from. Our research not only provides a different method that is based on abstractions but also accounts for their learning.

Third, this research provides a different theory of analogy that is based on device models and design patterns in which design patterns enable cross-domain transfer in design—that is, it proposes that the design knowledge that needs to be transferred between domains is the knowledge of high-level abstractions such as design patterns and not the specific designs in a source domain. This is not to claim that the specific designs will never get transferred to solve new problems, but rather to say that such transfer may occur only within a domain. In fact, this theory was built on model-based case adaptation (Goel, 1991a) and hence covers within-domain analogies also. This research provides the MBA process for analogy in which design patterns are learned from design analogues in a source domain, and are retrieved and instantiated in different target domains for solving design problems (both adaptive design and redesign). More specifically, this research shows how GTMs enable cross-domain transfer in the context of design by analogy and how GPPs enable cross-domain transfer in the context of redesign.

4. **Problem Reformulation:** The third dimension in our characterization of innovative design is the ability to reformulate design problems. Again, similar to the issues that concern non-local modifications and cross-domain transfer, there were three issues with respect to a theory of problem reformulation: (1) what is problem reformulation? (2) why is it necessary and important? and (3) how can it be enabled in device design?

First, by problem reformulation, we mean addition, deletion, or modification of constraints in a given problem after the process of design begins. Problem reformulation can be due to the specification of constraints by an external agent later in the design process or due to the discovery of constraints based on the evaluation of a design for the initial specification of the problem. We focused on the latter type of problem reformulation.

Second, problem reformulation is necessary because some constraints on a design are not uncovered until after the design is evaluated, for example, evaluated by its use in a real environment. The constraints that are generally discovered late in the process are about interactions of a design with its environment. The discovery of such constraints is late because the conditions of the environment may not be known completely at the beginning of the process. Even if the conditions were known, they might change from the time of initial problem specification to the time of design realization. When the environments are thus dynamic, designs may fail, although they satisfy the constraints of initial problem specifications. Then redesigning the failed design involves interpreting (or understanding) the design failures and identifying new constraints to be solved (Prabhakar and Goel, 1992).

Third, this research provides a theory of problem reformulation in which design patterns enable understanding of design failures (given as external feedback on evaluation of designs) by forming causal explanations. The design failures are specified in terms of undesired behavioral states and undesired behavioral state transitions using the SBF language. We described the MBA process in which GPPs can be retrieved and instantiated in the context of a design to understand the design failures by forming causal explanations for the failures. In this process, the causal explanations then facilitate the discovery of new constraints and thus enable reformulation of the design problem to include the discovered constraints.

5. **Acquisition of Design Patterns:** One of the important aspects of our theory of innovative design is that it not only accounts for the use of design patterns to enable innovation, but also accounts for their acquisition. The questions then were (1) what may be the origin of design patterns (i.e., what may be the knowledge from which the design patterns can be acquired), (2) why is acquisition necessary and important to a theory of innovative design, and (3) how can they be acquired?

In general, there may be different origins for the knowledge of design patterns: direct acquisition from a teacher, direct acquisition from natural language descriptions, learning from experience alone, learning from experience and interaction with a teacher under failures. We explored how design patterns can be acquired from design experiences including feedback from an oracle upon problem-solving failure. The task of learning design patterns in this research is really a discovery task because there is no assumption of a priori knowledge of the target concepts learned.

The acquisition of design patterns is important and necessary in a theory of innovative design for two reasons. First, a theory that accounts not only for the use of a proposed type of knowledge but also for its acquisition is more interesting and it accepts more constraints than one which cannot account for both learning and use. Second, cross-domain transfer is an important aspect in our characterization of innovative design. Our theory of cross-domain transfer is abstraction-based, in particular, it is based on design patterns, and under some conditions, successful transfer depends on the acquisition of design patterns.

This research provides a process account of how design patterns can be acquired from design experiences under different interaction conditions for feedback from an oracle upon problem-solving failure. In particular, it provides model-based methods for learning two types of design patterns (i.e., GPPs and GTMs) incrementally by abstraction over regularities in design experiences. This research proposes specific answers to issues in learning by abstraction from experience: device models together with the problem-solving context in which learning occurs suggest "what" to abstract from experiences and device models together with similarities and differences in experiences suggest the level of abstraction. Because of the type of knowledge captured in design patterns, we argued in this research that their acquisition from design experiences requires knowledge of device models in the

design experiences. More specifically, this research shows how SBF models of devices provide the content for abstracting design patterns and also provide constraints on that process.

Since GPPs can be viewed as concepts and GTMs as strategies, and both can be learned from design experiences, our learning theory covers both concept learning and strategy learning.

6. **Acquisition of SBF Models of Devices:** Since in our theory of innovative design, several processes are model-based in that they make use of qualitative models of specific devices, the acquisition of device models is as important as that of design patterns. In this research qualitative device models are represented in the SBF language. There are three questions here just like in the case of the acquisition of design patterns: (1) what may be the origin of device models? (i.e., what may be the knowledge from which device models can be acquired) (2) why is their acquisition necessary and important to a theory of innovative design? and (3) how can they be acquired?

Like design patterns, device models may also have different origins such as direct acquisition from a teacher, direct acquisition from natural language descriptions, learning from experience alone, learning from experience and interaction with a teacher under failures. We explored the last possibility while some previous work (Goel, 1989) on which this research is built explored learning from experience alone and some contemporary work in a related project called KA (Peterson et al., 1994) explored the direct acquisition of device models from natural language descriptions. Similarly, some previous work in qualitative device modeling (Bylander, 1991) has also considered the generation of new device models by a composition of primitive behaviors (i.e., consolidation).

Some reasons for why the acquisition of device models is important and necessary in a theory of innovative design are same as those for the importance of acquisition of design patterns. For instance, a theory is better constrained if it accounts for both acquisition and use of a particular type of knowledge. In addition, it is a fundamental question in qualitative device modeling and addressing that is an important task because our theory of innovative design relies on the availability (or acquisition) of specific device models.

Now the question is how might the SBF models of new designs be acquired automatically? In general, there can be different methods for the acquisition of new SBF models, partly depending on the origin of the SBF models. A few methods are (1) composition (or consolidation) of behaviors of primitive structural elements (Bylander, 1991), (2) revision of known models of similar devices (Goel, 1991b), (3) a combination of model revision and composition of behaviors of primitive structural elements, and (4) instantiation of design patterns in the models of known devices. In this research we took an adaptive modeling approach and proposed methods (3) and (4). The basic model-revision method is inherited

from the work of Goel (1991b) on which methods (3) and (4) are built.

7. **Organization, Indexing, and Index Learning:** Since we take a memory-based view of design, the questions regarding organization and indexing of different types of knowledge in memory are very relevant. Similarly, since our theory of design accounts for learning of new knowledge, the question about learning indices for the new knowledge also became important. In particular, we addressed the question of how a new design analogue and a design pattern might be indexed and organized in memory for later use, and how their indices can be acquired dynamically.

In general, there may be different ways of organizing the memory of design analogues and indexing them. In this research we took a task-directed approach and indexed the design analogues by functions and structural constraints, because the analogues were used for the task of design that involves specifying functional and structural constraints as input. However, within a given indexing scheme, there can be selection on the specific features that are used to index the designs. That is, for instance, the choice may be between indexing by all the features in the input specification of a design (i.e., exhaustive indexing) and indexing by only some features in the input specification that are important to the solution (i.e., selective indexing). This research provides an empirical comparison on the effectiveness of these two indexing schemes for retrieval of design analogues. The result is that the selective indexing is more efficient than the exhaustive indexing for retrieval of designs if the selection of indices is based on device models.

Similarly, within a given indexing scheme, the organization of designs can differ: flat organization *vs.* hierarchical organization. In the hierarchical organization scheme, design analogues are actually organized in multiple, parallel hierarchies because (1) design analogues are indexed both by functions and structure and (2) both function and structure consist of multiple features and not just a single feature. This research provides an empirical comparison on the effects of these two organization schemes for retrieval of designs under different indexing schemes. The result is that the hierarchical organization under selective indexing is more efficient for retrieval than the hierarchical organization under exhaustive indexing. Further, the latter itself is more efficient than the flat organization.

Thus this research provides an account of how design analogues are stored in memory. For design patterns too, we took a task-directed approach for indexing them. That is, GPPs are indexed by their behavioral abstractions while GTMs are indexed by the patterned functional differences because GPPs are used for explaining failures (i.e., undesired states and behaviors) and GTMs are used for achieving adaptation goals (i.e., reduce functional differences between a candidate function and a desired function). In this work, since there were not many GPPs and GTMs, we did not organize them hierarchically. This research also provides an account of how these different types of knowledge might interact in the context of design tasks.

For all the different types of knowledge acquired, i.e., design analogues and design patterns, this research provides model-based methods for acquiring their indices and organizing the knowledge in memory dynamically. In particular, this work shows that the SBF language provides an organization and indexing scheme for storing design analogues and design patterns in memory.

8. **Retrieval of Design Analogues and Design Patterns:** In a memory-based view of design, just like indexing and organization of knowledge are important, their retrieval is an equally important and complementary issue. The specific questions we addressed are how design analogues and design patterns can be retrieved from memory and what features in design problems determine their retrieval.

The retrieval mechanisms in a theory are, of course, closely tied with the indexing and organization of the knowledge. This research provides a mechanism for the retrieval of design analogues from a memory indexed in multiple ways and organized in multiple, parallel hierarchies. For the issue of what features might determine the retrieval of analogues, different theories have different answers: some suggest that only surface features of the problem matter (Ross, 1989; Gentner, 1989) while some others suggest that deep, semantic features of the problem are also equally important (Holyoak and Koh, 1987; Hintzman, 1986, Gentner, Ratterman and Forbus, 1993); yet others indicate that both surface and deep features may have their role but under different conditions (cf. this thesis). In our theory, the retrieval of design analogues is primarily based on partial match between specific features in the design problems (i.e., surface features) and the problems of design analogues. In addition, we also explored how deep, semantic features in terms of primitive functions may determine the retrieval when such features are available in the input or when they can be easily inferred. Since initial problem specifications may not refer to deep features, semantic retrieval at that stage of design would require an additional inference to go from surface features to semantic features. However, when the retrieval of a design is necessary while analyzing a source design analogue (an already retrieved design) and the SBF model of the source design (perhaps a subdesign is being sought in order to compose with the source design), the semantic features of the new desired design can be available from the model of the source design; and in such cases, semantic retrieval may be very useful.

This research also provides a method for the retrieval of design patterns from a memory indexed by behavioral abstractions (for GPPs) and patterned functional differences (for GTMs) and organized in a flat memory.

11.3 Model-Based Analogical Design

Although the overall goal of this research was to develop a theory of innovative design, since we characterized innovative design to include cross-domain transfer of design knowledge, this work also led to a theory of analogical design. In our theory of innovative design, high-level abstractions such as *design patterns* (in particular, those that capture functional and causal relationships among design elements) enable the three facets of design innovation, i.e., non-local modifications, cross-domain transfer, and problem reformulation.

Since in our theory of innovative design the design patterns can be learned from design experiences in one domain and get used to solve problems in a different domain, design patterns thus enable cross-domain transfer. We showed in particular how the two specific types of design patterns, namely, GPPs and GTMs, enable cross-domain transfer of design knowledge. Thus our theory of innovative design based on design patterns and SBF models of devices also leads us to a theory of model-based analogical design. This theory also accounts for model-based within-domain transfer of design knowledge in which the SBF models of devices constrain and guide the transfer process.

Our theory of model-based analogical design is in much contrast with and complementary to earlier theories of analogy that advocate direct transfer (i.e., transfer of a source analogue directly to a target problem irrespective of the domains they belong to) or abstraction-mediated transfer (i.e., transfer mediated by high-level abstractions but which nevertheless involves accessing a source analogue and transferring knowledge from it). Our theory is complementary to the earlier theories in three respects: first, it provides a different account of cross-domain transfer in which the transfer occurs from high-level abstractions to specific target problems; second, it not only advocates the use of high-level abstractions, but also accounts for their learning from experience in source domains; and third, it covers both within-domain transfer and cross-domain transfer in a seamless process.

BIBLIOGRAPHY

- Anderson, J. (1986). Knowledge compilation: The general learning mechanism. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II*. Morgan Kaufmann, Los Altos, CA.
- Ashley, K. and Rissland, E. (1988). A case-based approach to modeling legal expertise. *IEEE Expert*, 3(3):70-77.
- Barletta, R. and Mark, W. (1988a). Breaking cases into pieces. In *Proc. of the AAAI Workshop on Case-Based Reasoning*, pages 12-17.
- Barletta, R. and Mark, W. (1988b). Explanation-based indexing of cases. In Kolodner, J., editor, *Proc. of the DARPA Workshop on Case-Based Reasoning*, pages 50-60, San Mateo, CA. Morgan Kaufmann.
- Bassok, M. and Holyoak, K. (1989). Interdomain transfer between isomorphic topics in algebra and physics. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15:153-166.
- Bhatta, S. and Goel, A. (1992). Use of mental models for constraining index learning in experience-based design. In *Proceedings of the AAAI workshop on Constraining Learning with Prior Knowledge*, pages 1-10, San Jose, CA.
- Bhatta, S. and Ram, A. (1991). Learning indices for schema selection. In *Proc. of the Florida Artificial Intelligence Research Symposium*, pages 226-231, Cocoa Beach, FL.
- Birnbaum, L. and Collins, G. (1988). The transfer of experience across planning domains through the acquisition of abstract strategies. In Kolodner, J., editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 61-79, Clearwater Beach, FL.
- Brown, D. and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*. Pitman, London, UK.
- Bylander, T. (1991). A theory of consolidation for reasoning about devices. *International Journal of Man-Machine Studies*, 35(4):467-489.
- Bylander, T. and Chandrasekaran, B. (1985). Understanding behavior using consolidation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 450-454.
- Carbonell, J. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 137-161. Tioga, Palo Alto, CA.
- Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol. III*, pages 371-392. Morgan Kaufmann, Los Altos, CA.
- Catrambone, R. and Holyoak, K. (1989). Overcoming contextual limitations on problem-solving transfer. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15(6):1147-1156.

- Chandrasekaran, B. (1983). Towards a taxonomy of problem solving types. *AI Magazine*, 4(1):9-17.
- Chandrasekaran, B., Goel, A., and Iwasaki, Y. (1993). Functional representation as design rationale. *IEEE Computer*, pages 48-56.
- Cohen, W. (1988). Generalizing number and learning from multiple examples in explanation-based learning. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 256-269, Ann Arbor, MI. Morgan Kaufmann.
- Darden, L. (1991). Personal communication.
- de Kleer, J. and Brown, J. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24:7-83.
- DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145-176.
- Dietterich, T. G. (1992). Toward model-based learning: A case study in ecosystem prediction. In *Proceedings of the AAAI Workshop on Constraining Learning with Prior Knowledge*, pages 19-26, San Jose, CA.
- Dyer, M., Flowers, M., and Hodges, J. (1986). Edison: An engineering design system operating naively. In Sriram, D. and Adey, R., editors, *Proc. of the First International Conference on Applications of AI to Engineering Problems*, pages 327-342, Berlin. Springer.
- Elio, R. and Anderson, J. (1981). The effects of category generalizations and instance similarity on schema abstraction. *Journal of Experimental Psychology: Human Learning and Memory*, 7:397-417.
- Falkenhainer, B. (1990). A unified approach to explanation and theory formation. In Shrager, J. and Langley, P., editors, *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann, San Mateo, CA.
- Falkenhainer, B., Forbus, K., and Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1-63.
- Falkenhainer, B. and Michalski, R. (1986). Integrating quantitative and qualitative discovery: The abacus system. *Machine Learning*, 1:367-401.
- Fermi, E. (1937). *Thermodynamics*. Prentice-Hall, New York, NY.
- Flann, N. and Dietterich, T. (1989). A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187-226.
- Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85-168.
- Forbus, K. and Gentner, D. (1986). Learning physical domains: Toward a theoretical framework. In Carbonell, J. and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II*, pages 311-348. Morgan Kaufmann, Los Altos, CA.
- Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7:155-170.
- Gentner, D. (1989). The mechanisms of analogical reasoning. In Vosniadou, S. and Ortony, A., editors, *Similarity and Analogical Reasoning*, pages 199-241. Cambridge University Press, Cambridge, England.
- Gentner, D., Ratterman, M., and Forbus, K. (1993). The roles of similarity in transfer: Separating retrievability from inferential soundness. *Cognitive Psychology*, 25:524-575.

- Gero, J. (1990). Design prototypes: A knowledge representation schema for design. *AI Magazine*, 11(4):26-36.
- Gero, J., Tham, K., and Lee, H. (1991). Behaviour: A link between function and structure in design. In Brown, D., Waldron, M., and Yoshikawa, H., editors, *Preprints of the IFIP WG5.2 Working Conference on Intelligent CAD*, pages 201-230.
- Gick, M. and Holyoak, K. (1980). Analogical problem solving. *Cognitive Psychology*, 12:306-355.
- Gick, M. and Holyoak, K. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, 15:1-38.
- Goel, A. (1989). *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving*. PhD thesis, The Ohio State University, Department of Computer and Information Science, Columbus, Ohio.
- Goel, A. (1991a). A model-based approach to case adaptation. In *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 143-148, Chicago.
- Goel, A. (1991b). Model revision: A theory of incremental model learning. In *Proc. of the Eighth International Conference on Machine Learning*, pages 605-609, Chicago.
- Goel, A. (1992a). Recompositional analogy: A model-based approach to design reuse. In *Notes of the AAAI Spring Symposium on Computational Support for Incremental Modification and Reuse*, Palo Alto.
- Goel, A. (1992b). Representation of design functions in experience-based design. In Brown, D., Waldron, M., and Yoshikawa, H., editors, *Intelligent Computer Aided Design*, pages 283-308. North-Holland, Amsterdam, Netherlands.
- Goel, A. and Callantine, T. (1991). A control architecture for run-time method selection and integration. Technical report, Georgia Institute of Technology, College of Computing, Atlanta, GA. Also in *Procs. of the AAAI-91 Workshop on Cooperation Among Heterogenous Intelligent Systems*.
- Goel, A. and Callantine, T. (1992). An experience-based approach to navigational route planning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Raleigh, NC.
- Goel, A., Callantine, T., Shankar, M., and Chandrasekaran, B. (1991). Representation, organization, and use of topographic models of physical spaces for route planning. In *Proceedings of the Seventh Conference on Artificial Intelligence Applications*, Miami, FL. IEEE Computer Society Press.
- Govindaraj, T. (1987). Qualitative approximation methodology for modeling and simulation of large dynamic systems: Applications to a marine steam power plant. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(6):937-955.
- Greiner, R. (1988). Learning by understanding analogies. *Artificial Intelligence*, 35:81-125.
- Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, MA.
- Hammond, K. and Hurwitz, N. (1988). Extracting diagnostic features from explanations. In Kolodner, J., editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 169-178, San Mateo, CA. Morgan Kaufmann.
- Hammond, P. H. (1958). *Feedback Theory and Its Applications*. The English Universities Press Ltd., London, UK.
- Hayes, P. (1979). Naive physics manifesto. In *Expert Systems in the Microelectronics Age*, pages 242-270. Edinburgh University Press, Edinburgh, UK.

- Hayes, P. (1985). Naive physics i: Ontology for liquids. In *Formal Theories of the Commonsense World*, pages 71–107. Ablex, Norwood, NJ.
- Hinrichs, T. R. (1988). Towards an architecture for open world problem solving. In Kolodner, J., editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 182–189, San Mateo, CA. Morgan Kaufmann.
- Hinrichs, T. R. (1991). *Problem Solving in Open Worlds: A Case Study in Design*. PhD thesis, Georgia Institute of Technology, College of Computing, Atlanta, GA. Tech Report #GIT-CC-91/36.
- Hinrichs, T. R. (1992). *Problem Solving in Open Worlds: A Case Study in Design*. Erlbaum.
- Hintzman, D. (1986). "schema abstraction" in a multiple-trace memory model. *Psychological Review*, 93:411–428.
- Holyoak, K. (1984). Analogical thinking and human intelligence. In Sternberg, R., editor, *Advances in the Psychology of Human Intelligence*, Vol. 2, pages 199–230. Lawrence Erlbaum, Hillsdale, NJ.
- Holyoak, K. (1985). The pragmatics of analogical transfer. In Bower, G., editor, *The Psychology of Learning and Motivation*, Vol. 19. Academic Press, San Diego, CA.
- Holyoak, K. and Thagard, P. (1989a). Analogical mapping by constraint satisfaction. *Cognitive Science*, 13:295–355.
- Holyoak, K. and Thagard, P. (1989b). A computational model of analogical problem solving. In Vosniadou, S. and Ortony, A., editors, *Similarity and Analogical Reasoning*, pages 242–266. Cambridge University Press, Cambridge, England.
- Huhns, M. and Acosta, R. (1988). Argo: A system for design by analogy. *IEEE Expert*, 3(3):53–68.
- Huhns, M. and Acosta, R. (1992). Argo: An analogical reasoning system for solving design problems. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design*, Vol. II, pages 105–143. Academic Press, San Diego, CA.
- Iwasaki, Y. and Simon, H. (1986). Causality of device behavior. *Artificial Intelligence*, 29:3–32.
- Kambhampati, S. (1993). Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*, 10(2).
- Karp, P. (1989). *Hypothesis Formation and Qualitative Reasoning in Molecular Biology*. PhD thesis, Stanford University, Department of Computer Science, Stanford, CA.
- Karp, P. (1990). Hypothesis formation as design. In Shrager, J. and Langley, P., editors, *Computational Models of Scientific Discovery and Theory Formation*, pages 275–317. Morgan Kaufmann, San Mateo, CA.
- Keane, M. (1987). On retrieving analogues when solving problems. *Quarterly Journal of Experimental Psychology*, 39A:29–41.
- Kedar-Cabelli, S. (1985). Purpose-directed analogy. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 150–159, Irvine, CA.
- Kedar-Cabelli, S. (1988). Toward a computational model of purpose-directed analogy. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning II: An Artificial Intelligence Approach*, pages 284–290. Morgan Kaufmann, Los Altos, CA.
- Kling, R. (1971). A paradigm for reasoning by analogy. *Artificial Intelligence*, 2(2):147–178.

- Kolodner, J. (1984). *Retrieval and Organization Strategies in Conceptual Memory: A Computer Model*. Erlbaum, Hillsdale, NJ.
- Kolodner, J. (1993). *Case-Based Reasoning*. Morgan-Kaufmann, San Mateo, CA.
- Kolodner, J. L. (1989). Selecting the best case for a case-based reasoner. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*.
- Kolodner, J. L. and Simpson, R. L. (1984). Experience and problem solving: A framework. In *Proceedings of the Sixth Conference of the Cognitive Science Society*, pages 239–243.
- Kolodner, J. L. and Simpson, R. L. (1989). The mediator: Analysis of an early case-based problem solver. *Cognitive Science*, 13(4):507–549.
- Koton, P. (1988). Integrating case-based and causal reasoning. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 167–173, Northvale, NJ. Erlbaum.
- Kuipers, B. (1984). Commonsense reasoning about causality. *Artificial Intelligence*, 24:169–203.
- Laird, J., Newell, A., and Rosenbloom, P. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.
- Langley, P., Simon, H., Bradshaw, G., and Zytkow, J. (1987). *Scientific Discovery: An Account of the Creative Processes*. MIT Press, Boston, MA.
- Lebowitz, M. (1983). Generalization from natural language text. *Cognitive Science*, 7(1):1–40.
- Maher, M. and Zhao, F. (1987). Using experiences to plan the synthesis of new designs. In Gero, J., editor, *Expert System in Computer-Aided Design*, pages 349–369. North-Holland, Amsterdam.
- McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19:39–88.
- Minsky, M. (1975). A framework for representing knowledge. In Winston, P., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York.
- Mitchell, T. M., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80.
- Mittal, S. and Araya, A. (1992). A knowledge-based framework for design. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol. I*. Academic Press, San Diego, CA.
- Mittal, S., Dym, C., and Morjaria, M. (1986). Pride: An expert system for the design of paper handling systems. *Computer*, 19(7):102–114.
- Mostow, J. (1989). Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40:119–184.
- Mostow, J., Barley, M., and Weinrich, T. (1989). Automated reuse of design plans. *International Journal of Artificial Intelligence and Engineering*, 4(4):181–196.
- Mostow, J., Barley, M., and Weinrich, T. (1992). Automated reuse of design plans in bogart. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol. II*. Academic Press, San Diego, CA.
- Navinchandra, D. (1991). *Exploration and Innovation in Design: Towards a Computational Model*. Springer-Verlag, New York.
- Navinchandra, D., Sycara, K., and Narasimhan, S. (1991). A transformational approach to case-based synthesis. (*AI EDAM*), 5(1):31–45.

- Nersessian, N. J. (1995a). Abstraction via generic modeling in concept formation in science. In Cartwright, N. and Jones, M. R., editors, *Idealization in Science*. Editions Rodopi, Amsterdam.
- Nersessian, N. J. (1995b). Should physicists preach what they practice? constructive modeling in doing and learning physics. *Science and Education*, 4(3). Also to appear in M. Vincintini (Ed.), "Thinking Science for Teaching: The Case of Physics".
- Pazzani, M. (1991). Learning to predict and explain: An integration of similarity-based, theory-driven, and explanation-based learning. *The Journal of the Learning Sciences*, 1(2):153-199.
- Peterson, J., Mahesh, K., and Goel, A. (1994). Situating natural language understanding within experience-based design. *Int. J. Human-Computer Studies*, 41:881-913.
- Prabhakar, S. and Goel, A. (1992). Performance-driven creativity in design: Constraint discovery, model revision, and case composition. In Gero, J. and Sudweeks, F., editors, *Proceedings of the Second Int. Round-Table Conf. on Computational Models of Creative Design*, pages 101-128, Heron Island, Australia.
- Qian, L. and Gero, J. (1992). A design support system using analogy. In Gero, J., editor, *Proceedings of the Second International Conference on AI in Design*, pages 795-813, Pittsburgh.
- Rajamoney, S. (1988). *Explanation-Based Theory Revision: An Approach to the Problems of Incomplete and Incorrect Theories*. PhD thesis, University of Illinois, Department of Computer Science, Urbana, IL.
- Rajamoney, S. (1990). A computational approach to theory revision. In Shrager, J. and Langley, P., editors, *Computational Models of Scientific Discovery and Theory Formation*, pages 225-253. Morgan Kaufmann, San Mateo, CA.
- Ram, A. (1989). *Question-Driven Understanding: An Integrated Theory of Story Understanding, Memory, and Learning*. PhD thesis, Yale University, Department of Computer Science, New Haven, CT. Tech Report #710.
- Ram, A. (1990). Decision models: A theory of volitional explanation. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pages 198-205, Cambridge, MA.
- Ram, A. (1993). Indexing, elaboration and refinement: Incremental learning of explanatory cases. *Machine Learning*. In a special issue on case-based reasoning and learning. Also available as College of Computing, Georgia Tech, Tech Report #GIT-CC-92/03.
- Ram, A. and Cox, M. (1994). Introspective reasoning using meta-explanations for multistrategy learning. In Michalski, R. S. and Tecuci, G., editors, *Machine Learning: A Multistrategy Approach (Vol. IV)*, pages 349-377. Morgan Kaufmann, San Mateo, CA.
- Ram, A., Cox, M., and Narayanan, S. (1995). Goal-driven learning in multistrategy reasoning and learning systems. In Ram, A. and Leake, D., editors, *Goal-Driven Learning*. MIT Press, Bradford Books, Cambridge, MA.
- Rasmussen, J. (1985). The role of hierarchical knowledge representation in decisionmaking and system management. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:234-243.
- Rasmussen, J. (1987). Mental models and the control of actions in complex environments. Technical Report RISO-M-2656, RISO National Laboratory, DK 4000 Roskilde, Denmark.
- Reeves, L. and Weisberg, R. (1994). The role of content and abstract information in analogical transfer. *Psychological Bulletin*, 115(3):381-400.
- Riesbeck, C. and Schank, R. (1989). *Inside Case-Based Reasoning*. Erlbaum, Hillsdale, NJ.

- Ross, B. (1984). Reminders and their effects in learning a cognitive task. *Cognitive Psychology*, 16:371-416.
- Ross, B. (1987). This is like that: The use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13:629-639.
- Ross, B. (1989). Distinguishing types of superficial similarities: Different effects on the access and use of earlier problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15:456-468.
- Roverso, D., Edwards, P., and Sleeman, D. (1992). Machine discovery by model driven analogy. In Zytkow, J. M., editor, *Proceedings of the ML-92 workshop on Machine Discovery*, pages 87-97, Aberdeen, Scotland.
- Rychener, M., Farinacci, M., Hulthage, I., and Fox, M. (1992). Aladin: An innovative materials design system. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol. II*, pages 215-261. Academic Press, San Diego, CA.
- Schank, R. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, Cambridge.
- Schank, R. (1986). *Explanation Patterns: Understanding Mechanically and Creatively*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- Sembugamoorthy, V. and Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Riesbeck, editors, *Experience, Memory and Reasoning*, pages 47-73. Lawrence Erlbaum, Hillsdale, NJ.
- Shavlik, J. W. (1985). Learning about momentum conservation. In *Proceedings of IJCAI-85*, Los Angeles, CA.
- Shavlik, J. W. and DeJong, G. F. (1987). Bagger: An ebl system that extends and generalizes explanations. In *Proceedings of the Sixth National Conference on AI (AAAI-87)*, pages 516-520.
- Shinn, H. S. (1989). *A Unified Approach to Analogical Reasoning*. PhD thesis, Georgia Institute of Technology, School of ICS, Atlanta, GA.
- Simmons, R. and Davis, R. (1987). Generate, test and debug: Combining associational rules and causal models. In *Proceedings of the IJCAI-87*, pages 1071-1079, San Mateo, CA. Morgan Kaufmann.
- Simon, H. (1969). *The Sciences of the Artificial*. The MIT Press, Cambridge, MA.
- Stallman, R. and Sussman, G. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135-196.
- Stroulia, E. (1994). *Failure-Driven Learning as Model-Based Self-Redesign*. PhD thesis, Georgia Institute of Technology, College of Computing, Atlanta, GA.
- Stroulia, E. and Goel, A. (1992). Generic teleological mechanisms and their use in case adaptation. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 319-324, Bloomington, IN.
- Stroulia, E., Shankar, M., Goel, A., and Penberthy, L. (1992). A model-based approach to blame-assignment in design. In Gero, J., editor, *Proceedings of the Second International Conference on AI in Design*, pages 519-537, Pittsburgh.
- Sussman, G. and Steele, G. (1980). Constraints: A language for expressing almost-hierarchical descriptions. *Artificial Intelligence*, 14:1-39.

- Sycara, K. and Navinchandra, D. (1989). Integrating case-based reasoning and qualitative reasoning in engineering design. In Gero, J., editor, *Artificial Intelligence in Engineering Design*, pages 232-250. Computational Mechanics, UK.
- Sycara, K. and Navinchandra, D. (1991). Index transformation techniques for facilitating creative use of multiple cases. In Gero, J., editor, *Proceedings of the IJCAI-91 Workshop on AI in Design*, pages 15-20, Sydney. University of Sydney.
- Sycara, K. and Navinchandra, D. (1992). Retrieval strategies in a case-based design system. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol. II*, pages 145-163. Academic Press, San Diego, CA.
- Thagard, P., Holyoak, K., Nelson, G., and Gochfeld, D. (1990). Analog retrieval by constraint satisfaction. *Artificial Intelligence*, 46:259-310.
- Tong, C. (1992). Using exploratory design to cope with design problem complexity. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design, Vol. II*, pages 287-332. Academic Press, San Diego, CA.
- Tong, C. and Sriram, D. (1992). *Artificial Intelligence in Engineering Design, Vols. I-III*. Academic Press, San Diego, CA.
- Vosniadou, S. and Ortony, A. (1989). *Similarity and Analogical Reasoning*. Cambridge University Press, Cambridge.
- Williams, B. (1991). Interaction-based design: Constructing novel devices from first principles. In *Procs. of the IFIP International CAD Conference*.
- Wills, L. M. and Kolodner, J. L. (1994). Towards more creative case-based design systems. In *Proceedings of AAAI-94*, Seattle, WA.
- Winston, P. (1975). Learning structural descriptions from examples. In Winston, P., editor, *The Psychology of Computer Vision*. McGraw-Hill Book Co., New York, NY.
- Winston, P. (1980). Learning and reasoning by analogy. *Communications of the ACM*, 23(12):689-703.
- Winston, P. (1982). Learning new principles from precedents and exercises. *Artificial Intelligence*, 19(3):321-350.
- Winston, P. (1986). Learning by augmenting rules and accumulating censors. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II*, pages 45-61. Morgan Kaufmann, Los Altos, CA.
- Wisniewski, E. and Medin, D. (1991). Harpoons and long sticks: The interaction of theory and similarity in rule induction. In D.H. Fisher, M.J. Pazzani, and P.W. Langley, editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*. Morgan Kaufmann, San Mateo, CA.
- Zhao, F. and Maher, M. (1992). Using network-based prototypes to support creative design by analogy and mutation. In Gero, J., editor, *Proceedings of the Second International Conference on AI in Design*, pages 773-793, Pittsburgh.
- Zytkow, J. (1987). Combining many searches in the fahrenheit discovery system. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 281-287, Los Altos, CA. Morgan Kaufmann.

VITA

Sambasiva Bhatta is currently a Member of Technical Staff at NYNEX Science & Technology in White Plains, NY. From September, 1989 to June, 1995, he was a graduate student and also a Graduate Teaching and Research Assistant in the College of Computing at Georgia Institute of Technology. During the summer of 1991, he worked as a Summer Intern at the Human Interface Technology Center, NCR Corp. (now AT & T Global Information Solutions), Atlanta, GA. He obtained his MS in Information and Computer Science from the Georgia Institute of Technology in June, 1992. Before his graduate study at Georgia Tech, Bhatta received his Master of Technology degree in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, India, in 1988, and worked as a Research Engineer at the same institute during 1988-1989. Prior to that, he received his Bachelor of Technology degree in Electronics and Communications Engineering from the Regional Engineering College at Warangal, India in 1986. He was born in Pedana, Andhra Pradesh, India, on March 14, 1965.

Bhatta's areas of interest include Knowledge-Based Systems, Artificial Intelligence and Cognitive Science (in particular, problem solving, learning, and memory, and analogy, case-based reasoning, and model-based reasoning); Computer-Aided Engineering Design, Design Support and Design Education; and Intelligent Multimedia Interfaces. His primary objective is to work in applied research, and design and develop knowledge-based systems for applications in problem-solving tasks such as planning, design, diagnosis, and scheduling. At NYNEX Science & Technology, he is currently working on the Arachne/Athena project that involves automated route planning and design in the domain of telecommunication networks. His other interests include collecting stamps and coins, sketching, photography, and traveling.

one
To appear as a chapter in
Issues in Case-Based Design.
Maher and Pu (editors),
MIT Press, 1996.

KRITIK: AN EARLY CASE-BASED DESIGN SYSTEM

ASHOK K. GOEL, SAMBASIVA R. BHATTA, ELENI STROULIA
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
Tel: (404) 853-9371, (404) 853-9381
Fax: (404) 853-937
Email: {goel,bhatta,eleni}@cc.gatech.edu

Abstract: In the late 1980s, we developed one of the early case-based design systems called Kritik. Kritik autonomously generated preliminary (conceptual, qualitative) designs for physical devices by retrieving and adapting past designs stored in its case memory. Each case in the system had an associated structure-behavior-function (SBF) device model that explained how the structure of the device accomplished its functions. These case-specific device models guided the process of modifying a past design to meet the functional specification of a new design problem. The device models also enabled verification of the design modifications. Kritik2 is a new and more complete implementation of Kritik. In this paper, we take a retrospective view on Kritik. In early papers, we had described Kritik as integrating case-based and model-based reasoning. In this integration, Kritik also grounds the computational process of case-based reasoning in the SBF content theory of device comprehension. The SBF models not only provide methods for many specific tasks in case-based design such as design adaptation and verification, but they also provide the vocabulary for the whole process of case-based design, from retrieval of old cases to storage of new ones. This grounding, we believe, is essential for building well-constrained theories of case-based design.

1. Introduction

Design is a very common, wide-ranging and open-ended activity. It includes not only the design of physical artifacts but also abstract artifacts, such as software interfaces, and conceptual artifacts such as causal explanations. It can vary from everyday to specialized, naive to expert, and routine to creative design. While design sometimes is original, even revolutionary, much of design is evolutionary in that new designs are generated by adapting old designs. Evolutionary design includes both variant design, in which new designs locally differ from old designs in the values of specific parameters of the design elements, and adaptive design, in which new designs locally differ from old designs in the specific design elements.

Case-Based Reasoning (CBR) is a cognitively inspired computational theory in which new decisions are made by retrieving and modifying the decisions made in similar situations encountered in the past, and new problems are solved by retrieving and modifying the solutions to similar, previously encountered, problems. CBR thus views decision making and problem solving as memory tasks in that the memory supplies an answer in the neighborhood of the right answer, an „almost right“ answer that need only be tweaked to get to the right answer (Riesbeck and Schank, 1989). The memory is not only rich but also dynamic since new cases with potential for future use may enter the memory.

In the late 1980s, Goel and Chandrasekaran at the Ohio State University developed one of the earliest case-based design systems called Kritik (Kritik in Sanskrit roughly means „designer.“) (Goel, 1989; Goel and Chandrasekaran, 1989a; Goel and Chandrasekaran, 1989b; Goel and Chandrasekaran, 1992). Kritik was an autonomous system that addressed Function--to--Structure design tasks in engineering domains. In particular, it generated, adapted and evaluated preliminary (conceptual, qualitative) designs for physical devices such as simple electrical circuits and heat exchange devices. The Kritik experiment showed that CBR provides a good process account of the variant and adaptive aspects of preliminary design. But it also raised a number of content and strategic issues that appear to occur in all case-based systems:

1. What might be the content, representation and organization of a case?
2. How might a case be indexed? What might be the indexing vocabulary? How might the case memory be organized?
3. How might a new problem be specified? What might be the problem specification language?
4. Given a new problem, how might a similar case be retrieved from the case memory? How might a probe for exploring the case memory be prepared? What kinds of features in the new problem determine the similarity?
5. Once a case has been retrieved from memory, how might it be modified to address the new problem? What knowledge might guide the modification?
6. How might the candidate solution for new problem be evaluated? What knowledge might enable the verification of the candidate solution for new problem?
7. What might happen if the verification fails?
8. If the verification succeeds, how might the new case be stored in memory for potential use in future? How might it be indexed in memory and how might the indices be acquired dynamically?

In their work on Kritik, Goel and Chandrasekaran developed a model-based framework for addressing some of these issues. The key idea was that evolutionary design involves not only through past design experiences (i.e., cases) but also through comprehension (i.e., models) of how the designs work. While the high-level processes of variant and adaptive design are largely case-based, the design models give rise to both the vocabulary and the strategies for addressing the different tasks in the case-based process. The specific hypothesis was that since preliminary design is a Function--to--Structure mapping, the inverse Structure--to--Function map of old designs may provide guidance in adapting an old design to achieve anew functional specification. The Structure--to--Function map of a device design in Kritik is specified as a Structure-Behavior-Function model. The SBF model of a device explicitly specifies the structure and the functions of the device as well as its internal causal behaviors that explain how the structure delivers the functions, how the device functions are composed from the functions of its structural components.

Kritik was one of the earliest systems to integrate case-based and model-based reasoning. Experiments with Kritik showed that the SBF

models not only give rise to model-based adaptation strategies for making local modifications to old designs (Goel, 1991a) but that they also give rise to model-based simulation strategies for verifying whether the new design achieves the functions desired of it (Goel, 1991b; Goel and Prabhakar, 1991), and, in addition, provide the vocabulary for indexing the design cases (Goel, 1992).

In the early 1990s at Georgia Institute of Technology, we reimplemented Kritik's theory in a new system called Kritik2 (from InterLisp-D/Loops on Xerox Lisp to Symbolics CommonLisp/CLOS on Symbolics Lisp machines) and reproduced the earlier experiments for a larger class of devices (Bhatta and Goel, 1992; Stroulia and Goel, 1992; Stroulia *et al.*, 1992). While Kritik designed simple electrical circuits and heat exchange devices, Kritik2 also designs electromagnetic devices and electronic circuits containing operational amplifiers. Also, while Kritik's case memory was non-hierarchical (or „flat“), Kritik2's design cases are organized multiple hierarchies. Experiments with Kritik2 show that the SBF device models not only provide the vocabulary for indexing design cases but that they also enable the learning of new indices in order to better index new designs in the case memory (Bhatta and Goel, 1995). This chapter describes Kritik2's integrated theory of adaptive design, using examples inherited from Kritik.

2. An Illustrative Example

Let us consider the problem of designing an electrical circuit that will produce light of intensity 20 lumens, when a circuit known to produce light of intensity 10 lumens is available in the case memory. The following two subsections use this illustrative example to introduce Kritik2's device models and its process model for case-based design.

2.1. DEVICE MODELS

Kritik2's structure-behavior-function (SBF) model of a device, explicitly represents (i) the function(s) of the device (i.e., the problem), (ii) the structure of the device (i.e., the solution), and (iii) the internal causal behaviors of the device. The internal causal behaviors express Kritik2's comprehension of how the device works: they specify how the functions of

the structural components of the device are composed into the device functions.

Structure: The structure of a device in the SBF language is expressed in terms of its constituent components and substances and the interactions between them. Figure 1(b) shows the specification of the structure of the red light bulb circuit illustrated in Figure 1(a). Components and substances can interact with each other *structurally* and *behaviorally*. For example, electricity can flow from battery to bulb only if they are structurally *connected*, and due to the function *allow* electricity of the switch that connects the battery and the bulb. For simplicity, we ignore the wires in the circuit in the rest of our discussion, assuming that the other components are connected directly.

Function: The function of a device in the SBF language is represented as a schema that specifies the input behavioral state of the device, the behavioral state it produces as output, and a pointer to the internal causal behavior of the design that achieves this transformation. Figure 1(c) illustrates the function of the electrical circuit. Both the input state and the output state are represented as *substance schemas*. The input state specifies that the substance electricity at location battery in the topography of the device (Figure 1(a)) has the property voltage and the corresponding parameter 2 volts. The output state specifies the properties intensity and color, and the corresponding parameters 10 lumens and red, of a different substance, light, at location bulb. Finally, the slot *by-behavior* points to the causal behavior that achieves the function of producing light.

In Kritik2's memory, the design cases and their associated SBF models are indexed by the *functions* delivered by the devices. Thus the existing electric circuit is indexed by the function illustrated in Figure 1(c). The functions, in turn, act as indices into the internal causal behaviors of the SBF model through their *by-behavior* slot.

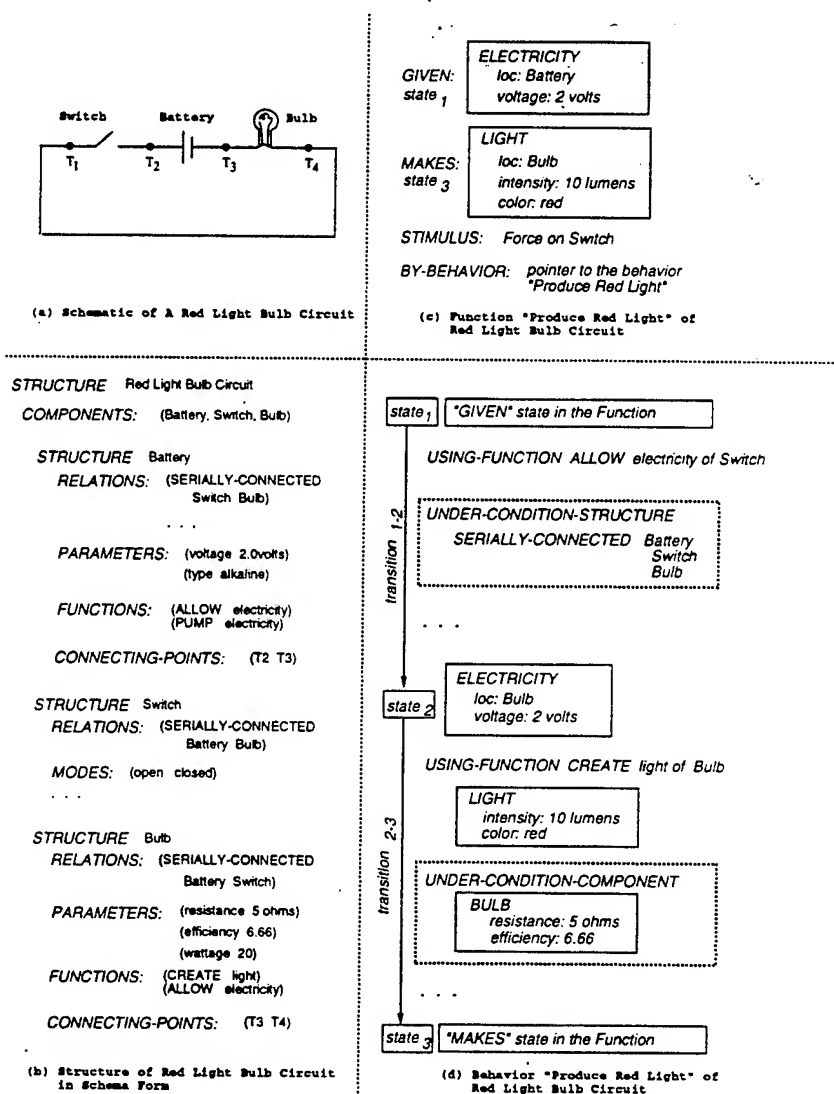


Figure 1: Design of A Red Light Bulb Circuit

Behavior: The SBF model of a device also specifies the internal causal behaviors that compose the functions of device substructures into the functions of the device as a whole. In the SBF language, the internal causal behaviors of a device are represented as sequences of *transitions* between *behavioral states*. The annotations on the state transitions

express the *causal*, *structural*, and *functional contexts* in which the state transitions occur and the state variables get transformed. The causal context specifies *causal relations* between the variables in preceding and succeeding states. The structural context specifies different *structural relations* among the components, the substances, and the different spatial locations of the device. The functional context indicates which functions of which components in the device are responsible for the transition. Figure 1(d) shows the causal behavior that explains how electricity in the battery is transformed into light in the bulb. State-1 describes the state of electricity at location battery and state-2 specifies the state of the same substance at location bulb; state-3 describes the state of a different substance, light, at location bulb. The annotation *using-function* on transition2->3 between state-2 and state-3 indicates that the transition occurs due to the primitive function *create light* of bulb. Similarly, the annotation *under-condition-structure* in transition1->2 specifies that the components battery, switch, and bulb need to be serially connected in order for the transition to occur.

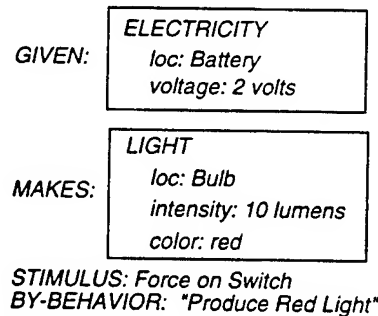


Figure 2: Function of the know red-light circuit

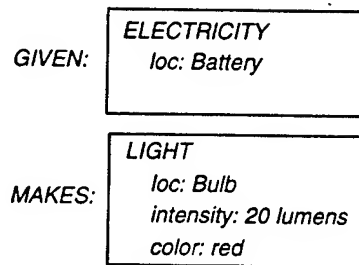


Figure 3: Desired function of a higher-intensity red-light circuit

Before we end this introduction to Kritik2's device models, it is useful to note that new problems are presented to Kritik2 in the SBF language. The specification of the new problem of designing an electric circuit producing light of higher intensity is shown in Figure 3. Note, that the two functions, the function of the known circuit (Figure 2) and the function of the desired one (Figure 3) are similar except for the value of the property intensity of the substance light at the output behavioral state.

2.2. COMPUTATIONAL PROCESS

Figure 4 illustrates Kritik2's processing. Given the problem of designing an electric circuit that will produce light of a specific intensity, the first step is to *retrieve a relevant design case* from memory. One issue here is how to index the design cases in memory and what features in the new problem to use to probe the case memory. New problems for Kritik2 are specified by the functions desired of a device, for example, the desired function of the new circuit illustrated in Figure 3. The cases are indexed by the functions delivered by the known designs, for example, the function delivered by the existing low-intensity circuit illustrated in Figures 1(c) and 2. The functional specification of the new problem is used as a probe into the case memory, and the cases whose functional specifications at least partially match the probe are selected. If more than one design is selected, then they are heuristically ordered by their „ease of adaptation“.

The second step in this process is to *adapt* the relevant parts of the design solution in the retrieved case to fit the new problem and the third step is to evaluate the modified design. Kritik2 interleaves these two steps. (The dotted lines between different steps in Figure 4 indicate that they are interleaved.) It first diagnoses the „faulty“ parts in the retrieved design so that the modified design satisfies the requirements of the new problem. That is, it views the retrieved design as having „failed“ to satisfy the requirements of the new problem, and „diagnoses“ and „repairs“ the failure. The important issue then becomes the identification of *what* to modify, i.e., the diagnosis of the „faulty“ parts in the retrieved design that need to be repaired in order to meet the functional specification of the new problem. In Kritik2's model-based approach, the device model indexed off the design case guides the localization and identification of the faulty parts that need to be repaired. In our example, the SBF model of the existing circuit (Figure 1(d)) suggests that the voltage of the battery is responsible for the difference in the functions of the known and the desired device (i.e., light intensity 10 lumens vs. 20 lumens). This results in the candidate modification of replacing the battery with a new one of higher voltage.

The next step in this process is to *evaluate* the candidate solution for the new problem, that is, to verify whether the proposed design satisfies the functional requirements of the new problem. In Kritik2, first, the changes due to the proposed modification of battery replacement are propagated to the other parts of the SBF model of the existing circuit (i.e., the substep of behavior modification). This results in a revised SBF model for the candidate design for the desired circuit (but without any structural changes as yet). Next, the revised SBF model is *qualitatively simulated* to verify if its causal processes result in the functions specified in the new problem (i.e., the substep of behavior verification). In our example, the simulation of the revised circuit model indeed results in the achievement of the function specified in the new problem. If the evaluation succeeds, then, in the repair step, the candidate modification is actually executed on the structure of the candidate design (i.e., the substep of structure modification). If the evaluation fails, then an alternative modification can be generated if possible. If an alternative candidate modification is not available, then an alternative candidate design (i.e., a different retrieved

case) can be selected for adaptation: In this way, Kritik2 interleaves the diagnosis, repair and evaluation steps:

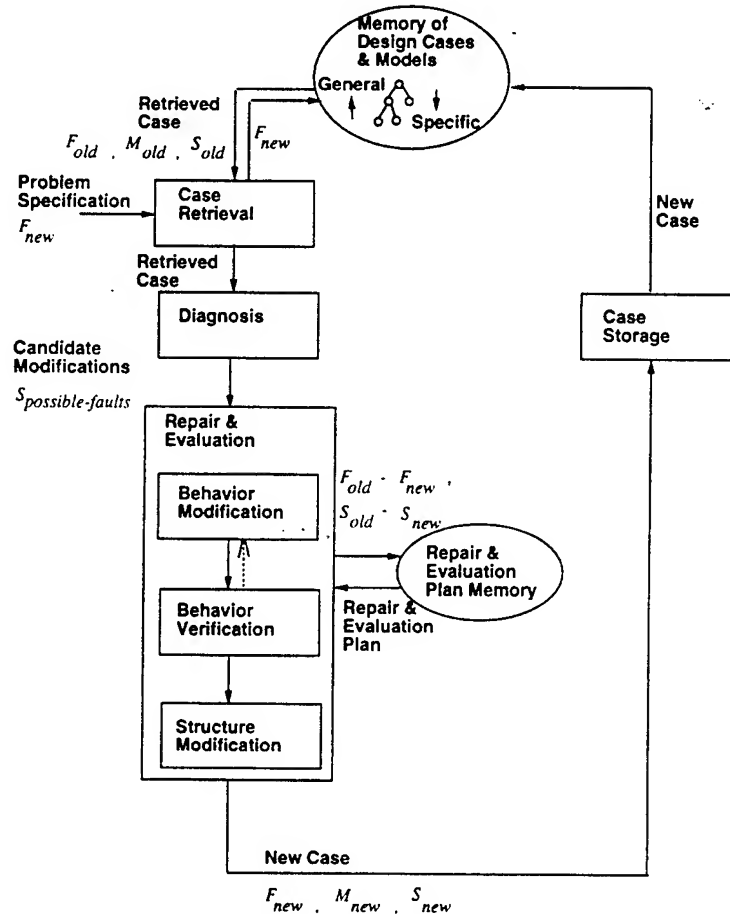


Figure 4: Kritik2's Process Model of Design

In the final step, the design that satisfies the requirements of the new problem is *stored* in the case memory potential reuse in future problem solving. In order for the new case to be useful in future, it needs to be stored in the „right“ place in memory, that is, its indices need to be selected appropriately. The device model of the new design case indicates what features of the problem specification are crucial in the functioning of

the new design, and thus the model helps learn the „right“ indices for the new case.

3. Structure-Behavior-Function Models

In this section, we specify the SBF language for representing design cases and device models using as examples the above electric circuit and a nitric acid cooler (NAC) (Figure 5), a simple device which cools nitric acid by exposing the pipes through which it flows to contact with cold water.

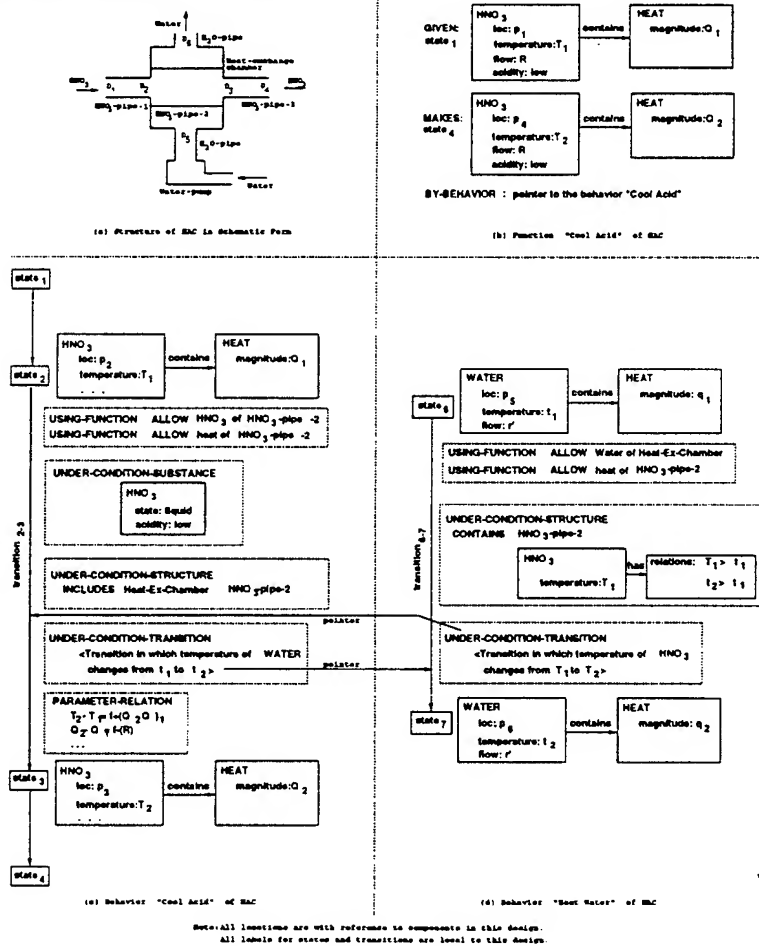


Figure 5: Design case of low-acidity nitric acid cooler

The schema for representing a design case in the SBF language (shown in Figure 6) consists of three slots: function, structure, and behaviors. The fillers for these slots are „coherent“ in that the behaviors specifies internal causal behaviors that explain how the structure delivers the function. Each slot in any SBF schema may be filled with one of three entities: an element from an enumerated set of primitive entities in the SBF language, another schema, or a list of other schemas.

design:

function: the intended output behavior of the device

behaviors: the internal causal behaviors of the device

structure: the device components and their structural relations

Figure 6: Design Case Schema

3.1. STRUCTURE

Figure 7 shows the schema for representing a design structure in the SBF language. The structure of the device is described hierarchically in terms of its constituent structural elements. The constituent elements of a device may be *primitive components* (i.e., the components assumed by Kritik2 to be primitives of the design domain), such as a battery for example. Alternatively, they may be *substructures*, such as an operational amplifier in an electronic circuit, which can be further described in terms of smaller constituent elements. Each structural element in this hierarchy (except for the overall structure of the device) points to the structural elements of which it is a part. In addition to the *part-of* relation, the structure schema can also specify the following structural relations: *containment* of a substance in a component or in another substance, *inclusion* of a component within another component, and *connection* between two components. Consider the example of the low-acidity nitric acid cooler (NAC) schematically shown in Figure 5(a). In this design, heat can flow from the nitric acid to the water only when the nitric-acid-pipe is *included* in the heat-exchange chamber. The connection between two components can be of two types: *serial* and *parallel*. The two connectivity relations differ in that the former specifies a relation between two components such that the output of one component becomes the input to the other, while the

latter specifies that the two components share the same input and same output.

structure:

components: a set of structural elements into which the element under description can be decomposed

structural-relations: a set of relations between the sub-elements of the structure under description

Figure 7: Structure Schema

3.2. BEHAVIORAL STATE

In the component-substance ontology of SBF device models, a behavioral state (input, output, or intermediate state) can be of two types: component state, which concerns the state of a component, and substance state, which concerns the state of a substance in the device. The component and substance behavioral states are characterized by the variables characteristic of the respective component and substance specified in the states. Figure 8 shows the schema for a behavioral state in the SBF language.

behavioral state:

previous: previous state

next: next state

enabled-by: preceding state-transition

enabling: succeeding state-transition

substance-state-schema: substance description at this state:

location: the location of the substance in the device

main-substance: the schema for the substance:

is-a : pointer to a prototype substance
(property value unit)*

{contained-substances: their description at the state }

OR

component-state-schema: component description at this state:

component: the schema for the component:

is-a : pointer to prototype component
(parameter value unit)*

mode: the mode of the component's operation at this state

Figure 8: Behavioral State Schema

The behavioral state schema contains links to previous and next states, preceding and succeeding state transitions, and either a component-state schema or a substance-state schema. The schema for a component state (partially) specifies the component under description, its mode of operation in the state, and points to the component schema. An example of a component state is the state of the switch in the electrical circuit when its mode is closed. The component schema itself specifies the type of component with the *is-a* slot, and a partial list of the component parameters along with their values. Similarly, the schema for a substance state (partially) specifies the state of the substance under description at a particular point in the device topology. It contains the slots for the *location* of the substance, and other substances contained in the main substance under description, and it points to the schema for the main substance. Again, the schema for a substance itself specifies the type of substance with the *is-a* slot, and a partial list of substance properties along with their values. We explain the representation of components and substances in detail a little later.

3.3. FUNCTION

Functions of devices can be of different kinds, such as transformation functions, control functions, maintenance functions, and prevention functions. Kritik2 currently deals with transformation functions only, in which the device transforms an input behavioral state into a different behavioral output state. Figure 9 shows the schema for the representation of transformation functions in the SBF language. In addition to the input and output behavioral states (*given* and *makes* respectively), the function schema contains the *by-behavior* slot for specifying the internal causal behavior that transforms the input state into the output state, the *stimulus* slot for specifying the interaction of the device with its environment that triggers its functioning, and the *provided* slot for specifying the environmental conditions necessary for the functioning of the device. An example of *stimulus* is force on switch, as shown in the functional specification of the electric circuit in Figure 1(c). In the specification of a desired function in a new design problem, the slot *by-behavior* would not be filled, and the slots *given*, *stimulus*, *provided* too may not be filled (see for example the specification of the new desired electric circuit in Figure

3). This schema for representing device functions is borrowed from Sembugamoorthy and Chandrasekaran 1986.

functional specification:

makes: output behavioral state
 {given: input behavioral state }
by: causal behavioral-state sequence
 {stimulus: event in the external environment triggering the
 functioning of the device }
 {provided: external to the device conditions necessary for the
 functioning of the device }*

Figure 9: Functional Specification Schema

Figure 5(b) shows the function „Cool Nitric Acid“ of the low-acidity NAC. Both the *given* and *makes* behavioral states of this function are *substance states*. The former specifies the state of in-flowing nitric acid, while the latter specifies the state of the nitric acid as it flows out of the device. In addition, the *by-behavior* slot points to the Cool Acid internal causal behavior that explains the above transformation.

Note that the function of a device in SBF models is an abstraction of the internal causal behaviors of the device. For transformation functions, the initial state and final state in an internal causal behavior are respectively the input state and output state in the function. For instance, the given and make states of the function of nitric acid cooler (Figure 5(b)) are respectively the same as the initial state (*state-1*) and the final state (*state-4*) in the internal causal behavior Cool Acid of the nitric acid cooler (Figure 5(c)). Note also that the functions of a device in SBF models are a subset of the set of its observable, output behaviors. In particular, a function is an output behavior of the device intended by the designer. For instance, the abstraction of the internal causal behavior „Heat Water“ of the nitric acid cooler (Figure 5(d)) is an output behavior of the device. But, in SBF models, this output behavior is included under the device functions only if it was actually intended by the designer.

3.4. BEHAVIORAL STATE TRANSITION

A behavioral state transition is a partial description of a transformation of one behavioral state into another during the functioning of the device.

Figure 10 shows the schema for representing such a transition in the SBF language. In addition to the links to the previous and next states, the behavioral state transition schema contains the slots *by-behavior*, *using-function*, *as-per-domain-principle*, *parameter-relations*, and *conditions* of different kinds that have to be true in order for the transition to occur.

state-transition:

previous-state: preceding state

next-state: succeeding state

{ by-behavior: pointer to a more detailed behavior explaining the transition }

{ using-function: component's function }*

{ as-per-domain-principle }*

{ parameter-relations }*

{ condition }*

Figure 10: Behavioral State Transition Schema

A behavioral transformation of a device element may be explained at several levels of causal abstraction and structural aggregation. Thus, the specification of a state transition may include a pointer to another behavior (through the *by-behavior* slot) which explains in greater detail the transformation described by that transition. The *by-behavior* pointer results in a hierarchical organization of the device internal behaviors. In addition to pointing to a more detailed behavior, a state transition may be explained in terms of the functions of structural elements of the device (i.e., *using-function* slot), or in terms of a domain principle (i.e., *as-per-domain-principle* slot) such as physics laws (e.g., the law of conservation of momentum). The *using-function* slot of a behavioral state transition schema is filled with a list of schemas each of which refers to a component in the device and a primitive function of that component.

Moreover, the transition schema may be annotated with qualitative equations (i.e., *parameter-relations* slot) describing the changes to the values of different substance properties and component parameters because of the transition. Qualitative equations may be based on physics principles but are specific to the device parameters. The *parameter-relations* slot of the state-transition schema is filled with a list of qualitative equations, where each qualitative equation itself consists of a qualitative relation between values of two substance properties or between

values of a substance property and a component parameter. A qualitative relation is an enumerated type and can have one of the two values: *directly-proportional-to* and *inversely-proportional-to*.

Often the occurrence of a state transition is conditioned upon the co-occurrence of other behavioral states in the device (the representation of this condition by a pointer to the state via *under-condition-state*), or the co-occurrence of other state transitions (a pointer to the transition via *under-condition-transition*), or specific structural relations among the device elements (a list of structural relations via *under-condition-structure*), or specific property values of a substance (a pointer to the partial description of the substance via *under-condition-substance*), or specific parameter values of a component (a pointer to partial description of the component via *under-condition-component*). The SBF language provides the vocabulary for specifying all five different types of conditions in a state transition.

For instance, in *transition2->3* (Figure 5(c)), the annotation *under-condition-substance* specifies that the behavior *allow* of nitric-acid-pipe-2 can allow the flow of only some substances: the substance should be in liquid state and should have low acidity (i.e., behavioral or causal context). Further, the annotation *under-condition-structure* specifies the structural relation that Heat-Exchange-Chamber *includes* nitric-acid-pipe-2 in order for the transition to occur (i.e., structural context). Annotations may also include conditions on other transitions as indicated by *under-condition-transition*. For example, *transition2->3* refers to *transition6->7* in another behavior (the behavior „Heat Water“) of NAC shown in Figure 5(d). In addition, a transition may be annotated by knowledge of deeper domain principles and qualitative equations as indicated in Figure 5(c).

3.5. BEHAVIOR

An internal causal behavior is a sequence of alternating behavioral states and behavioral state transitions. Figure 5(c) shows a fragment of the causal behavior that explains how nitric acid is cooled from temperature T1 to T2. *State-2* is the preceding state of *transition2->3* and *state-3* is its succeeding state. *State-2* describes the state of nitric acid at location p2 and so does *state-3* at location p3. The different types of annotations on

transition2->3 indicate the different types of causal contexts under which the transition can occur. For example, the annotation *using-function* in *transition2->3* indicates that the transition occurs due to the primitive function *allow* of *nitric-acid-pipe-2*.

3.6. OTHER KNOWLEDGE

In addition to design cases and their associated device models, Kritik2 has knowledge of the primitive functions, and the generic components and substances of the design domain. While Kritik2's SBF models are case-specific, its knowledge of components and substances is generic, i.e., case-independent. But all domain knowledge, from case-specific SBF models to generic components and substances, is represented in the same SBF language. In addition, the different types of knowledge are cross-indexed in Kritik2 as indicated in Figure 11. For instance, design cases index into their SBF device models, and a SBF device model is described in terms of the participating substances, components, and primitive functions. The partial specification of a substance or a component in the SBF model contains pointers to the more complete specifications of generic substances and components as indicated in the discussion of the behavioral state schema above.

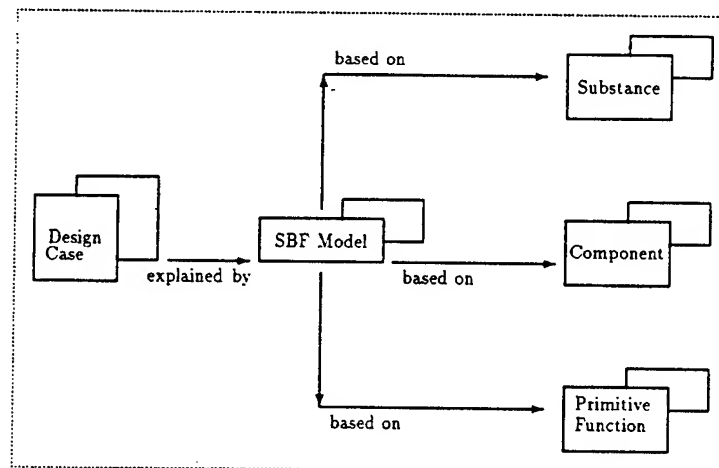


Figure 11: Types of Knowledge in Kritik2

A primitive function in the component-substance ontology of SBF models can be one of the following: *allow*, *pump*, *create*, and *destroy*. This typology of primitive functions is directly borrowed from Bylander (1991).

Unlike a device, a primitive component is a structure assumed to be non-decomposable by Kritik2. A component is represented as a schema consisting of the slots: *is-a*, *structural-relations*, *parameters*, *modes*, *functions*, and *connecting-points*. Figure 12 shows a portion of Kritik2's memory of primitive components organized in a taxonomic hierarchy. The schema for a specific component, *battery-1*, is shown in the dashed box. *Is-a* is a pointer through which a specific component is linked to a prototypical component in Kritik2's conceptual memory of components. For instance, *battery-1 is-a* battery in Figure 12. The *structural-relations* slot specifies a list of structural relations between this component and the others in the device. The *modes* slot in a component schema specifies one or more modes of operation of the component. The *parameters* slot contains a list of characteristic parameters of the component and their corresponding values and units. For example, the volume of a heat-exchange chamber has a value of 1 cu. ft. The *functions* slot of a component contains the set of primitive functions that the component delivers and the *connecting-points* slot specifies the structural points in the component where the other components can be connected.

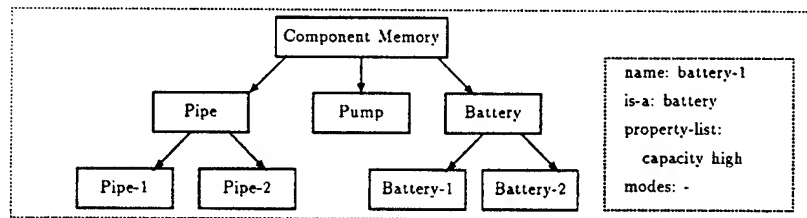


Figure 12: A Snapshot of Kritik2's Component Memory

Figure 13 shows a portion of Kritik2's conceptual memory of substances organized in a generalization-specialization hierarchy. The schema for representing a specific substance nitric acid is shown in the dashed box. *Is-a* is a pointer through which a specific substance is linked to a more general substance. For instance, nitric-acid *is-a* liquid in Figure

13. *PropertyList* contains a list of characteristic properties of the substance and the corresponding values and units. For example, the temperature of nitric acid in a specific behavioral state of a given device may have a value of T1 degrees. In the description of a substance in a behavioral state of a specific device, only some of these characteristic properties are of interest, and thus only their values are specified. Note that in Figure 13 many properties for a substance nitric acid are shown as „---“ which means that Kritik2 knows that these properties are relevant to this substance but in its generic knowledge there are no values specified for these properties. However, a specific substance in a specific device, for instance, nitric acid in low-acidity NAC may have additional values specified (i.e., acidity: low; temperature: T1) as shown in Figure 5(b).

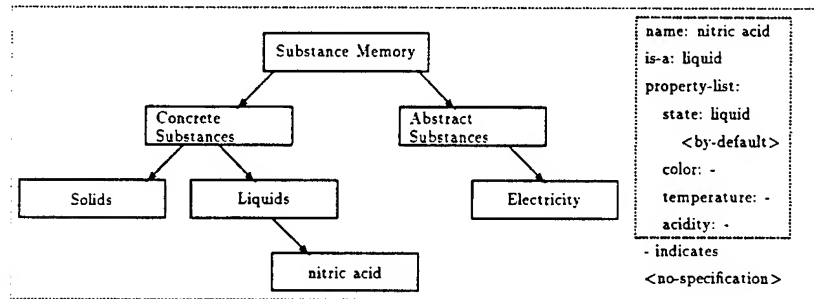


Figure 13: A Snapshot of Kritik2's Substance Memory

4. Case-Based Adaptive Design

Figure 4 illustrates Kritik2's computational process for case-based adaptive design. In this section, we describe how the SBF models give rise to the vocabulary and strategies for addressing the different subtasks of case-based adaptive design.

In the domain of physical devices, a typical problem in preliminary design is to design a device that achieves specific functions. This Function--to--Structure design task takes as input a specification of the functions desired of a device, and has the goal of giving as output a specification of a structure that delivers the desired functions. Consider,

for instance, the task of designing a *sulfuric acid cooler*, a device that delivers the function of cooling sulfuric acid of high acidity. Kritik2 accepts representations of new problems in the SBF language. The specification of the desired function in the SBF language is shown in Figure 14.

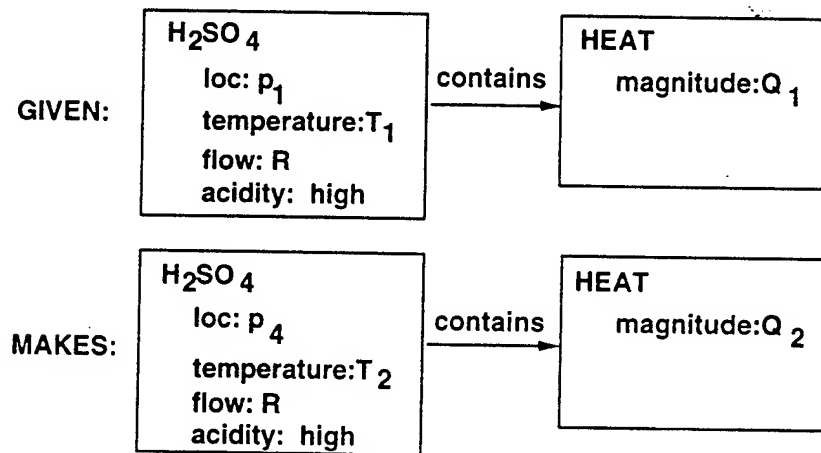


Figure 14: Function of cooling high-acidity sulfuric acid

4.1. CASE RETRIEVAL

The task of case retrieval takes as input (i) the functional specification of a desired design, and (ii) the functional specifications of design cases stored in memory. It has the goal of giving as output an ordered set of known design cases that can potentially be adapted to satisfy the functional specifications of the desired design. The retrieved cases are ordered by a qualitative estimate of their ease of adaptation for satisfying the functional specification of the desired design. The computational advantage of retrieving and ordering a *set* of known designs is that if adaptation of one design fails, then another design can be selected without again probing the case memory.

4.1.1. Organization and Indexing of the Case Memory

The retrieval of appropriate design cases raises the issues of indexing the design cases, matching the design problem with the problems in past

design cases, selecting a set of candidate design cases (when there are many „partial matches“), and finally ordering the selected designs. The case indexing in Kritik2 is task-specific: since Function--to--Structure design problems are specified by the functions desired of the new device, the stored design cases in Kritik2 are indexed by their functions. The SBF language provides the vocabulary for representing the functions delivered by the stored designs.

The design cases are organized in a generalization-specialization hierarchy. The properties of the substances specified in the device functions are used as dimensions along which the designs are generalized/specialized. For example, designs of *acid coolers* are organized along the dimension of property *acidity* and discriminated on the corresponding parameters low vs. high, as shown in Figure 15. The property *acidity* is important because the choice of *pipe* in the design depends on whether it has to allow a low-acidity substance or a high-acidity substance. The exact dimensions of generalization depend on the past design experiences of Kritik2. The nitric-acid cooler case (see Figure 15) is the design of the low-acidity nitric acid cooler that we described in the last section and it is stored under the category of low-acidity coolers.

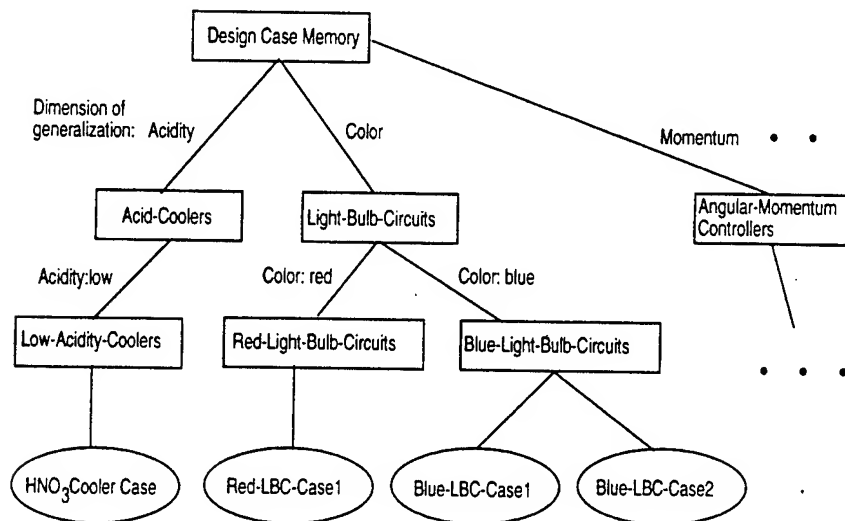


Figure 15: A Snapshot of Kritik2's functionally organized case memory

The functions at the higher-level nodes in this hierarchy are more general than those at the lower-level nodes of the hierarchy. That is, along a single dimension defined by a particular property, the values of this property in the functions of designs associated with a higher-level node subsume the values of this property in the functions of cases associated with a lower-level node. For instance, the higher-level node acid-coolers in Figure 15 has both classes of designs, low-acidity coolers and high-acidity coolers, associated with it. In contrast, the lower-level node low-acidity-coolers has only the designs of acid coolers with low acidity. Formally, the set of cases associated with a node in the hierarchy is a superset of the set of cases associated with any of its immediate child nodes.

4.1.2. Case Retrieval: Selection of Known Designs

The task of case retrieval is decomposed into the subtasks of *selection* and *ordering*. The selection subtask takes as input (i) the functions specified in the new problem, and (ii) the functions delivered by the stored cases. It has the goal of giving as output a set of known design cases that can be potentially modified to satisfy the requirements of the new problem specification.

The selected cases are such that the differences between their functions and the functions in the new problem are of a type that Kritik2 knows how to reduce. The SBF vocabulary for representing device functions gives rise to a taxonomy of functional differences, i.e., a taxonomy of differences between two functions. For example, two functions may differ only in their input states, or only in their output states, or in both. Further, two substance states (input or output) from two different functions may differ in several ways: their substances can differ (i.e., substance difference), the values of a common substance-property can differ (i.e., substance-property-value difference), or a substance property may be specified only in one of the states (i.e., substance-property-unspecified difference and substance-property-additionally-specified difference). Kritik2 knows how to reduce substance and substance-property-value (for a single or multiple properties) differences between the output behavioral states of two designs. Thus it selects only these designs whose functions differ from the function in the new problem in these ways.

Input: • Desired function, $F_{desired}$.

Output: • A set of design cases Set_{Cases} whose functions partially match $F_{desired}$.

Assumptions: • **root-list** contains the root nodes of all the hierarchies along with the dimensions of generalizations, and **dimension-list** contains all the dimensions of generalization.

Procedure:

```

SELECT( $F_{desired}$ ):
begin
   $Set_{Cases} = \{\}$ ;
   $selected-nodes = \{\}$ ;
   $tobe-seen-nodes = \{node_i \mid node_i \in root-list \wedge$ 
    function associated with  $node_i$  and  $F_{desired}$ 
    have at least one common property specified.  $\}$ 
  while not-empty( $tobe-seen-nodes$ ) do
    begin
       $child-nodes = MATCHING-CHILDREN(F_{desired}, first(tobe-seen-nodes))$ ;
      if empty( $child-nodes$ )
        then  $selected-nodes = first(tobe-seen-nodes) \cup selected-nodes$ ;
        else  $tobe-seen-nodes = tobe-seen-nodes \cup child-nodes$ ;
       $tobe-seen-nodes = rest(tobe-seen-nodes)$ ;
    end;
   $Set_{Cases} = \{case_i \mid node_i \in selected-nodes \wedge case_i \text{ is associated with } node_i, \}$ ;
  return ( $Set_{Cases}$ );
end.

MATCHING-CHILDREN( $F_{desired}$ ,  $node-in-hierarchy$ ):
begin
   $child-nodes = \{\}$ ;
  foreach ( $dimension \in dimension-list$ ) do
    begin
       $child = DISCRIMINATE(dimension, F_{desired}, get-children(dimension, node-in-hierarchy))$ ;
      if not-empty( $associated-cases(child)$ )
        then  $child-nodes = \{child\} \cup child-nodes$ ;
    end;
  return ( $child-nodes$ );
end.

DISCRIMINATE( $property, F_{desired}, children$ ):
begin
  foreach ( $child \in children$ ) do
    begin
      if ( $value-of(property, input-state(F_{desired})) = value-of(property, input-state(F(child)))$  )
         $\vee (value-of(property, output-state(F_{desired})) = value-of(property, output-state(child)))$  )
        then return( $child$ );
    end;
  return ( $nil$ ); /* as failure */
end.

```

Figure 16: The Selection Algorithm

Figure 16 shows the algorithm that Kritik2 uses for selecting design cases. It searches through the functionally organized case memory along the dimensions of generalization that correspond to the properties specified in the desired function. Along each dimension of generalization, it goes as far specific as possible comparing the value of that property specified in either the input state or the output state of the desired function. It collects the design case associated with the most specific value it could reach along each dimension. Note, that in this and subsequent algorithms, the notation

slot-name(schema) denotes the value of a slot in a schema. For example, PropertyList(State-IN(F)) denotes the value of the slot PropertyList in the schema for State-IN which in itself is a slot of the schema for function F.

For example, the matching of the functional specification of the high-acidity sulfuric acid cooler (Figure 14) with the functions of the designs stored in the case memory (Figure 15) results in the selection of the design for the low-acidity nitric acid cooler (Figure 5). Consider a hypothetical situation where the memory contained both the low-acidity nitric-acid cooler and a neutral-acidity motor-oil cooler (i.e., a design for cooling motor oil which has neutral acidity). In such a situation, Kritik2 would have selected both these cases because both of them match the desired function partially on the property of acidity.

4.1.3. Case Retrieval: Ordering of Selected Designs

Several situations are possible on the retrieval of a set of stored cases relevant to a given problem:

1. *Exact Match*: The set of selected design cases includes a design whose functional specification exactly matches that of the desired design. In this situation, the structure of the exactly matching design is the solution, and problem solving is terminated.
2. *No Match*: The set of selected design cases is empty. In this situation, no solution to the design problem by the case-based method, and problem solving can be terminated (or, in principle, some other method for the design task can be selected).
3. *Single Partial Match*: The set of selected design cases contains no exactly matching design and exactly one partially matching design. In this situation, there is no need for ordering the selected design cases.
4. *Multiple Partial Matches*: The set of selected design cases contains no exactly matching design and more than one partially matching designs. In this situation, there is a need for ordering the selected design cases for further processing.

As mentioned above, for the sulfuric-acid cooler example the selection subtask results in a single partial match because the design of the nitric acid cooler is the only stored design whose functional specification partially matches with the desired function of high-acidity sulfuric-acid cooler. But, for the purpose of illustrating Kritik2's method for ordering

cases, consider a hypothetical situation where two low-acidity cooler designs, the low-acidity nitric-acid cooler and a low-acidity sulfuric-acid cooler were selected.

The ordering subtask takes as input (i) the desired design specification, and (ii) a set of selected design cases and has the goal of giving as output the same set of cases ordered according to some measure of their ease of adaptation. In Kritik2, the ease of adaptation of a partially matching design is measured by the qualitative „distance“ between the function it delivers (which represents the current state in the design adaptation space), and the function desired of it (which is the goal state). There are two aspects to the estimation of this distance between the goal state from the current state:

1. *Behavioral States*: The distance of the goal state from the current state depends on the degree of match between the functional specification of the desired design and that of the candidate design case.

For instance, if the input behavioral state in the function of the desired design exactly matches the input behavioral state in the function of the retrieved design, while the output behavioral states match only partially, then the distance between the goal state and the current state is smaller than if both the input and output states were to match only partially. Similarly, if both the input and output behavioral states in the function of the desired design partially match with the input and output behavioral states in the function of the retrieved design respectively, then the current state is closer to the goal state than if only the input states match partially or if only the output states match partially.

2. *Behavioral State Features*: the distance of the goal state from the current state depends on which features in the function of the candidate design case match with the corresponding features in the function of the desired design. That is, the distance depends on which features in the input and output behavioral states in the function specification of the candidate design need to be *transformed* to match with the corresponding features in the input and output behavioral states in the functional specification of the desired design.

For example, it is in general easier to transform the value of a property of a substance into another value than to transform one substance into another. Thus, if the function of one partially matching design

differs from that of the desired design in the value of some property only, while the function of another partially matching design differs in the substance itself, then the former design is probably closer to the desired design than the latter. Similarly, it is in general easier to transform the value of one property of a substance into another value than to transform values of two properties, and so on. Of course, these *heuristics* for determining the distance between the goal state and the current state can only provide an estimate, and not an accurate measure, of the distance between them. Also, these heuristics can be *domain specific*.

In sum, Kritik2's heuristic estimate for case ordering is based on how many of the input and output behavioral states, and which state features and how many of them, match in the functional specification of the new problem and the function delivered by the candidate design. In the hypothetical situation where both a low-acidity nitric-acid cooler and a low-acidity sulfuric-acid cooler were selected, Kritik2 would order the latter as a better match than the former because the sulfuric acid cooler matches with desired function on the *substance* also.

4.2. DESIGN ADAPTATION

The task of design adaptation involves, first, the mapping of the differences between the desired function of the new problem and the function delivered by the candidate design into potential modifications to the structure of the candidate design (*functional differences* => *structural modifications*), and second, the evaluation and execution of the candidate modifications. The generation of useful candidate structural modifications is computationally complex because the differences between the function desired of and delivered by the retrieved design can be large and many, the needed structural modifications can be large and many, there may be no simple correspondence between the functional differences and the structural modifications, and the needed structural modifications can interact with one another and with the components in the structure of the known design.

Kritik2 decomposes the adaptation into the subtasks of diagnosis and repair. It views the retrieved design as a „faulty“ solution to the new problem and „repairs“ it appropriately. Solving the diagnosis and repair tasks effectively and efficiently requires knowledge that constrains and

focuses the processes of identifying possible faults and generating potentially useful modifications. In this section we will discuss how the SBF model of the retrieved design provides Kritik2 with the needed knowledge.

4.2.1. Diagnosis

The task of „diagnosis“ in the context of adaptive design is to identify possible „faults“ in the known design that, if fixed, can help to deliver a solution to the new problem. The diagnosis task takes as input (i) the new desired function, (ii) a retrieved design, and (iii) the differences between the desired function and the function delivered by the retrieved design. It has the goal of giving as output a set of plausible faults in the retrieved design. Each possible fault is described as a three-tuple consisting of either a substance or a component in the retrieved design, a property of that substance or component, and a relation between that property and some substance or component property in the output state of the desired function.

The SBF model of the known design specifies how the internal causal mechanisms of the device compose the functional abstractions of its structural components into the functions of the device as a whole. Thus, Kritik2 uses that knowledge to identify the causes that prevent the solution of the retrieved case to satisfy the new problem. In particular, it uses the algorithm shown in Figure 17, which given the functional differences and the SBF model of the known design traces through the model and identifies specific structural elements (components or substances) that can potentially be modified in a way that will result in the accomplishment of the function desired of the new design.

Input:

- Desired function, F_{new} .
- Source design case, C .
- Functional difference, $FD = F_{new} - F_{old}$, where F_{old} is the function in C .

Output:

- A set of possible faults (or candidate modifications), $S_{possible-faults}$.

Procedure:

initialize

$F_{old} = \text{functional-spec}(C)$;

$S_{possible-faults} = \{(E, P_i, \text{Relation}) \text{ where}$
 $E \in \{\text{Sub, Comp}\}, P_i \in \text{PropertyList}(E), \text{ and}$
 $\exists P' \in \text{PropertyList}(\text{StateOVT}(F_{new})) \text{ s.t. } P' \text{Relation } P_i$
 $\text{and } P' \text{ is the property whose value needs to be changed,}$
 $\text{and } \text{Relation} \in \{\text{directly-proportional-to, inversely-proportional-to}\}\}$;

$B_{old} = \text{function-by-behavior}(F_{old})$;

begin

$S_{possible-faults} = \text{BACKTRACE}(\text{behavior-final-state}(B_{old}), S_{possible-faults})$;

end.

BACKTRACE (state, $S_{possible-faults}$)

current-state = state

LOOP

previous-state = state-previous-state (current-state);

IF previous-state = NIL. **THEN** Exit **LOOP**

CASE :

(1) **IF** there is a qualitative equation in state-previous-transition(current-state),
and property $P_i \in S_{possible-faults}$ such that $P_i^{old} \text{Relation } P_i$, where
 P_i is a parameter of a component or a property of another substance E , and
 P_i^{old} is a property in the current design.
THEN $S_{possible-faults} = S_{possible-faults} \cup \{(E, P_i, \text{Relation})\}$
where $P_i^{new} = \text{Relation}^{-1}(P_i^{old})$, and
 P_i^{new} is the value of property P_i in the desired design.

(2) **IF** there is a functional abstraction of a component E (i.e., Comp in using-function)
and a condition on the substance properties on which E operates
(i.e., Conditions_{SUB}) in state-previous-transition(current-state),
and property $P_i \in S_{possible-faults}$ such that $P_i^{old} = P_i$, where
 P_i is a property of the substance in Conditions_{SUB} , and
 P_i^{old} is a property in the current design.
THEN $S_{possible-faults} = S_{possible-faults} \cup \{(E, P_i, \text{directly - proportional - to})\}$
where P_i^{new} determines the type of component E , and
 P_i^{new} is the value of property P_i in the desired design.

(3) **IF** there is a pointer to a new behavior sequence B' such that
the transition state-previous-transition(current-state)
depends on a transition, trans. of B'
THEN spawn
BACKTRACE (transition-next-state(trans.), $S_{possible-faults}$)

END-CASE

current-state = previous-state

goto **LOOP**

END-LOOP

Figure 17: The Diagnosis Algorithm

In our example, since the substance-property-value difference *low-acidity* \Rightarrow *high-acidity* occurs in the function of cooling low-acidity nitric acid, Kritik2 uses this function to access the internal causal behavior responsible for it, a fragment of which is shown in Figure 5(c) (recall that in the SBF model, functions act as indices to the behaviors responsible for them). Then, it traces through the retrieved, starting from the final state of the behavior and checking each state transition in the behavior to determine whether reducing the substance-property-value difference *low-*

acidity=>high-acidity requires any component in the transition to be modified (or replaced). If so, it identifies a potential fault, which could be eliminated by the corresponding structure modification.

For example, when Kritik2 arrives at the transition *transition2->3* shown in Figure 5(c), it finds that nitric-acid-pipe2 allows the flow of only low-acidity substances (i.e., some property of this component is related to the property *acidity* of the substances that it can allow). It therefore generates the structure modifications of (i) component-parameter adjustment (in case nitric-acid-pipe2 can allow the flow of high-acidity substances in a different parameter setting), (ii) component-modality change (in case nitric-acid-pipe2 can allow the flow of high-acidity substances in a different mode of operation), and (iii) component replacement (in case the first two modifications are not possible and nitric-acid-pipe2 has to be replaced with some sulfuric-acid-pipe2 which can allow the flow of high-acidity substances).

In this way, Kritik2 uses the SBF model for NAC to generate structural modifications that can help reduce the functional difference *low-acidity=>high-acidity*. Similarly, given the functional difference of *substance1=>substance2* (nitric acid=>sulfuric acid) Kritik2's diagnosis results in the generation of the structure modification of substance substitution *nitric acid=>sulfuric acid*.

4.2.2. Repair

The task of „repair“ is to execute a candidate modification given a set of possible modifications. In the context of adaptive design, this involves modifying the old design so that the possible faults are eliminated. It takes as input (i) the desired function, (ii) the retrieved design, (iii) the functional difference between the desired and the retrieved, and (iv) a set of possible faults as identified by the diagnosis task, and gives as output a modified model and its corresponding function.

As mentioned in section 4.1.2, the SBF language provides a vocabulary for expressing certain types of functional differences between design cases, such as substance difference, substance-property-value difference, substance-location difference, component difference, component-modality difference, and component-parameter difference. In

addition, it provides the vocabulary for expressing certain types of modifications to the structure of a design, such as substance substitution (including substance generalization and specialization), component modification (including component replacement, component-modality change, and component-parameter adjustment), relation modification (for example, series-to-parallel and parallel-to-series conversion), substructure deletion (for example, component deletion), and substructure insertion (for example, substructure replication).

Given the function desired of a design (e.g., to cool high-acidity sulfuric acid) and the function delivered by the selected design case (e.g., to cool low-acidity nitric acid), Kritik2 classifies the differences between the two functions into its typology of functional differences. If the desired and the delivered functions differ in more than one feature, then it heuristically ranks the differences in order of the difficulty of reducing them. In the NAC example, for instance, the desired function and the delivered function differ in two features: *substance1=>substance2* (*nitric acid=>sulfuric acid*), and *value1=>value2* of property *acidity* (*low-acidity=>high-acidity*). Since, in the domain of physical devices that can be modeled in terms of flow of substances between components, reducing the difference *substance1=>substance2* is in general less difficult than reducing *value1=>value2* of a property, Kritik2 reduces the latter before the former.

Given the substance-property-value difference between the retrieved design of low-acidity NAC and the desired function of cooling high-acidity sulfuric acid, Kritik2's ontology suggests that such a difference can potentially be reduced by the structural modifications of (i) component-parameter adjustment, (ii) component-modality change, and (iii) component replacement. Kritik2, prior to diagnosis however, does not know what components in the given design need to be modified. This is determined by the diagnosis of the NAC SBF model, which we have already seen in section 4.2.1.

In Kritik2 the repair step is interleaved with the evaluation step. This interleaving includes two further subtasks: (i) repair-and-evaluation plan selection, and (ii) repair-and-evaluation plan instantiation. During the former, the functional difference and the structure modification are used to

retrieve an applicable repair & evaluation plan from Kritik2's repair & evaluation plan memory. During the latter, the selected plan is applied *first* to the behavior of the known case. The structure of the known design is modified only after verifying (by simulation of the SBF model) that the modification will result in the achievement of the desired function.

4.3. EVALUATION OF THE CANDIDATE MODIFICATIONS

Since modifications in general are done to some localized parts of the solution in the known design case, it is essential to evaluate whether each initial modification, and the subsequent ones it may invoke in other parts of the device, are leading towards the satisfaction of the requirements of the new problem. An important element in Kritik2's integrated theory is that the SBF model of the known design can be modified and simulated to verify if the proposed modifications indeed result in a solution for the new problem without actually making modifications to the solution (i.e., the device structure in design problem solving). The task of repair & evaluation in Kritik2 takes as input (i) the functional specification of a desired design, (ii) the functional and structural specification of a candidate design, (iii) the SBF model of the candidate design, and (iv) the specification of a candidate modification to the structure of the candidate design. It has the goal of modifying the behavior of the retrieved design and giving as output an evaluation of whether the candidate modification, upon execution, would result in satisfying the functional specification of the desired design.

4.3.1. A Model-Based Method for Repair & Evaluation

The evaluation of a candidate modification, such as component replacement for example, involves two subtasks. The first task is to determine whether a design that satisfies the functional specification of the needed component (e.g., the functional specification of the sulfuric-acid-pipe, that is, a pipe which can allow high acidity substances) is available in memory. Recall that in the previous section, we described how Kritik2 determines that one candidate modification in adapting a low-acidity NAC to cool high-acidity sulfuric acid is the component replacement of nitric-acid-pipe2 with a component that allows high-acidity substances. If the needed design is not available in memory, then the candidate modification

of component replacement is not feasible. Thus, one subtask of repair & evaluation is to determine the feasibility of the candidate modification.

Let us suppose that there exists such a component as the one required by the modification plan. Then, the second subtask of repair & evaluation is to determine whether, upon execution, the candidate modification would result in satisfying the functional specification of the desired design. This involves the *simulation* of the effects of (the execution of) the candidate modifications on the output behaviors of the candidate design. The availability of the SBF model for the known design and the localization of the candidate modification gives rise to the method of *model revision*. In this method, the causal and output behaviors of the modified design are obtained by revising the causal and output behaviors of the known design (i.e., behavior modification). If the revision succeeds, i.e., if the revised causal behavior results in the desired output behavior of the desired design, then the candidate modification may be executed on the candidate design (i.e., structure modification). If the revision fails, then an alternative candidate modification may be evaluated.

The types of knowledge required for revising the SBF model of the known design depend on the type of candidate modification to be executed on it. For instance, the method for model revision corresponding to the candidate modification of component replacement requires knowledge of how to *compose causal behaviors*. More specifically, it requires knowledge about how to compose the causal behavior of the new component with causal behavior segments from the known design. For adaptation problems in which the „distance“ between the known and the desired design is „small“, the candidate modifications are local, and knowledge is available for revising the model for the candidate design, the method of model revision is a computationally attractive method for evaluating whether the behavioral effects of a structure modification would result in satisfying the behavioral specification of a given desired design (i.e., behavior verification).

Kritik2 uses a plan- and model-based method for the task of modification evaluation, that is, it uses repair & evaluation plans which in turn make use of the SBF model of the candidate design to perform the interleaved steps of repair and evaluation. A skeletal repair & evaluation

plan embodies knowledge of how the causal and output behaviors of the new design can be composed from the causal and output behaviors of the known design and the causal and output behaviors of other substructures such as components. The operations specified by such a plan are at two levels. At the first level, it specifies the tasks of design retrieval and model revision. At the second level, the *subplan* for model revision specifies the operations for modifying the causal and output behaviors of the known design.

The relation between the types of knowledge required for model revision and the types of candidate modification implies a family of model-revision plans, and, hence, a *family of repair & evaluation plans*, with particular methods applicable to specific candidate modifications. The need for a set of stored skeletal plans for the modification-evaluation task implies yet another subtask of the task, namely, the task of *selection* of the particular plan applicable to a given candidate modification.

Thus, in general, the plan-based method decomposes the task of repair & evaluation into three subtasks: *plan selection*, *design case retrieval*, and *simulation of behavioral effects*. The task of design case retrieval has been discussed in Section 4.1. The other two tasks can be characterized as follows.

1. *Plan Selection*: The task of plan selection takes as input the specification of a candidate modification, and has the goal of giving as output a specific repair & evaluation plan applicable to the given candidate modification. (Note again that the repair and evaluation are interleaved in our computational model.)
2. *Simulation of Behavioral Effects*: The task of simulation of behavioral effects takes as input (i) the function of the desired design, (ii) the SBF model of a candidate design, and (iii) the specification of a candidate modification. It has the goal of giving as output an evaluation of whether the effects of (the execution of) the candidate modification on the output behaviors of the known design result in satisfying the function of the desired design.

4.3.2. Selection of Repair & Evaluation Plans

The repair & evaluation plan memory contains a plan for each type of structural difference that corresponds to a structural modification. The stored plans are indexed by the types of structural modifications to which

they are applicable as well as the functional differences they can reduce. The retrieval of a plan applicable to a given structure modification is performed by the *associative method*. In this method, elementary structure modifications are directly mapped onto the stored plans.

Once a repair & evaluation plan corresponding to an elementary structure modification is retrieved, it can be instantiated and executed. This section describes one type of repair & evaluation plans that corresponds to substructure modification, i.e., substance-modification plans. The other types are component-modification plans, relation-modification plans, and substructure-deletion plans. The discussion assumes the availability of the causal behaviors of the known design, and, in particular, the specific causal behavior to be revised (as determined by the task of localization of structure modifications, a subtask of diagnosis.)

Substance-Modification Plans: In this type of structure modifications, a substance in the known design case is *substituted* by another substance. The substance-substitution plan does not involve design case retrieval as a subtask. Instead, model revision is performed directly. For instance, if the design problem is to modify the design of the low-acidity NAC to cool high-acidity sulfuric acid (Figure 18), the substitution of nitric acid by high-acidity sulfuric acid is generated as a candidate modification. Then the causal behavior for the SAC is obtained by a substitution of nitric acid by high-acidity sulfuric acid in the causal behavior segments one of which was shown in Figure 5(c).

Also, in the example of designing a high-acidity SAC, the *under-condition-transition* pointer in *transition2->3* is used to retrieve the causal behavior „Heat Water“ of the NAC. This behavior is traced to find the pointer to the behavior segment of cooling nitric acid (Figure 5). The pointers are modified to point to the corresponding behavioral states and state transitions in the behavior segments of the behavior „Cool Sulfuric Acid.“

Finally, the schema for the desired output behavior (i.e., the function) of the high-acidity SAC (shown in Figure 18) is revised. A pointer to the causal behavior „Cool Sulfuric Acid“ composed above is placed in the *by-behavior* slot of the function. The other slots are filled by copying their values from the schema for the function of the NAC shown in Figure 5.

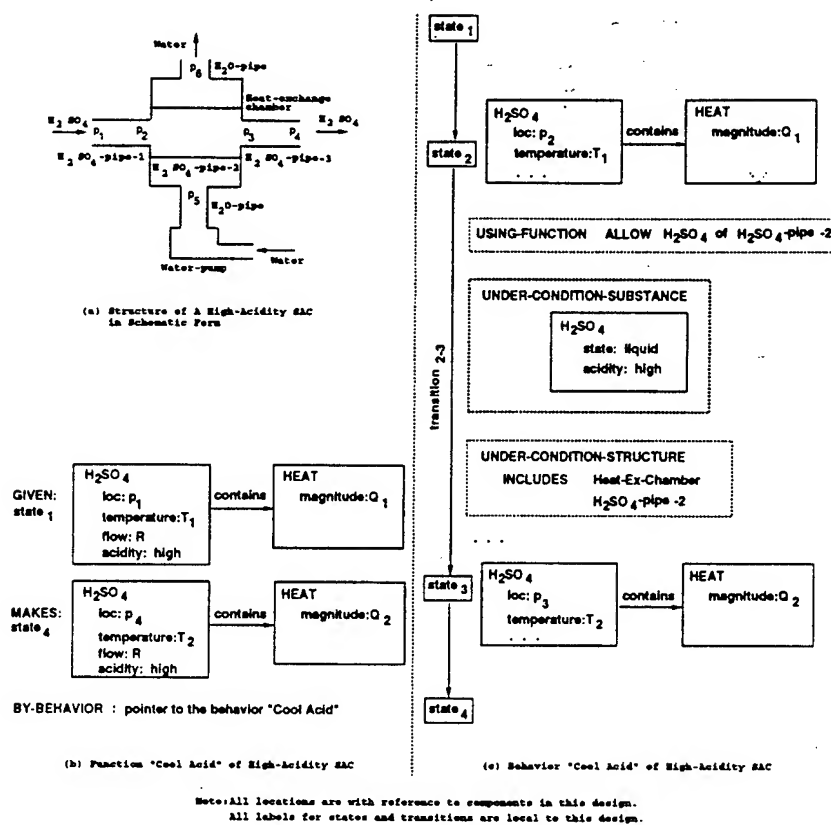


Figure 18: Design case of high-acidity sulfuric acid cooler

4.3.3. Simulation of Behavioral Effects

This task involves instantiating a repair & evaluation plan in the localized behavior, propagating those changes to the other parts of the same behavior and to the dependent behaviors, and finally simulating the behavior (from the input state to the output state) to determine if the revised model results in the achievement of the desired function. Kritik2 instantiates the retrieved plan in the context of the SBF model of the known case, and executes it on the model to produce an SBF model for the new design.

The model-revision process (pointed to by the retrieved repair & evaluation plan) specifies a compiled sequence of abstract operations.

Figure 19 shows the algorithm that Kritik2 uses for revising the model of an old design and evaluating the proposed modifications by simulating the behavioral effects (i.e., the substeps of behavior modification and behavior verification). The function *update* in the algorithm updates the value of a given property in a given state or a given qualitative equation. At the end of *simulate*, comparing the initial and final states of the new modified behavior with those in the desired function verifies whether the modifications worked.

To illustrate the model-revision process, let us return to the example of designing the high-acidity sulfuric acid cooler (SAC) shown in Figure 18. Recall that the retrieval task resulted in the selection of the design for the nitric acid cooler (NAC), where the two devices differ in that (i) while SAC cools high-acidity sulfuric acid, NAC cools low-acidity nitric acid, and (ii) while the pipes through which sulfuric acid flows in SAC need to allow high-acidity liquids, the pipes through which nitric acid flows in NAC allow only low-acidity liquids. The first of these two differences, *nitric acid=>sulfuric acid*, is an instance of the *substance substitution* type of structural differences; the second, *pipe(allow low-acidity substances)=>pipe(allow high-acidity substances)*, is an instance of the *component replacement* type of structural differences. The structures of SAC and NAC differ in more than one way, and, in the class of domains of interest, revising a SBF model to accommodate the structural difference of component replacement is in general more difficult than revising it to accommodate the difference of substance substitution. Therefore, Kritik2 ranks the two differences between the structures of SAC and NAC so that the model for NAC is first revised for the difference of *pipe(allow low-acidity substances)=>pipe(allow high-acidity substances)*, and then for the difference of *nitric acid=>sulfuric acid*.

Input: • Behavior in new design, B_{new} , in which a chosen property from the diagnosis has been changed.
 • Model of the old design, M_{old} .
Output: • Model of the new design, M_{new} , modified to be consistent with the changes in B_{new} .
Procedure:
 initialize
 $S_{diffs} = \{ (P_i, P_{i, value}^{old}, P_{i, value}^{new}), \text{ where } P_i \text{ is a property whose value } P_{i, value}^{old} \text{ in state } state_i^{old} \text{ has changed to } P_{i, value}^{new} \text{ in state } state_i^{new} \text{ and } state_i^{old} \text{ and } state_i^{new} \text{ are respectively from } B_{old} \text{ and } B_{new}. \}$
 SIMULATE ($state_i^{new}, S_{diffs}$)
 SIMULATE ($state, S_{diffs}$)
 current-state = state
 LOOP
 next-state = next-state (current-state);
 IF next-state = NIL. THEN Exit LOOP
 CASE :
 (1) IF P_i is mentioned in next-state
 THEN update(P_i , next-state, current-state)
 (2) IF there is a qualitative equation qc in state-next-transition(current-state), such that $P_i = f(P_i)$ where $(P_i, P_{i, value}^{old}, P_{i, value}^{new}) \in S_{diffs}$ and P_i is mentioned in next-state
 THEN update(P_i, qc)
 $S_{diffs} = S_{diffs} \cup \{ (P_i, P_{i, value}^{old}, P_{i, value}^{new}) \}$
 (3) IF there is a functional abstraction of a component E (i.e., Comp in using-function) and a condition on the substance properties on which E operates (i.e., $Conditions_{SUB}$) in state-previous-transition(current-state) such that $P_i = P_i$, where P_i is a property of the substance in $Conditions_{SUB}$, and $(P_i, P_{i, value}^{old}, P_{i, value}^{new}) \in S_{diffs}$
 THEN update($P_i, Conditions_{SUB}$)
 $S_{diffs} = S_{diffs} \cup \{ (P_i, P_{i, value}^{old}, P_{i, value}^{new}) \}$
 (4) IF there is a pointer to a new behavior sequence B' such that the transition state-next-transition(current-state) affects a transition $trans$ of B'
 THEN spawn
 SIMULATE (transition-prev-state($trans$), S_{diffs})
 END-CASE
 current-state = next-state
 goto LOOP
 END-LOOP

Figure 19: The Model Revision Algorithm

Let us consider the revision of the model for NAC, given the structure difference of *pipe(allow low-acidity substances) => pipe(allow high-acidity substances)* between the structures of NAC and SAC. This structural difference can suggest as one of the candidate repair & evaluation plans the one that corresponds to *component-replacement*. The model-revision process for component-replacement revises the SBF model for NAC in several steps. First, from the specification of the old pipe (nitric-acid-pipe2) in NAC, it determines that the nitric-acid-pipe2 plays a functional role in *transition2->3* of the behavior „Cool Nitric Acid“ of NAC. Then, it decomposes the behavior into three segments: (i) the transition in which the old component (nitric-acid-pipe2) plays a

functional role (in the present example, *transition2->3* in the behavior „Cool Nitric Acid“ shown in Figure 5(c), (ii) the sequence of state transitions preceding it (not fully shown here), and (iii) the sequence of state transitions succeeding it (also not fully shown here). Next, the transition in which the old component (nitric-acid-pipe2) plays a role is revised by replacing the functional abstraction of nitric-acid-pipe2 (which allows the flow of low-acidity liquids) by the behavioral abstraction of the new pipe (sulfuric-acid-pipe2) in SAC (which allows the flow of high-acidity liquids). Then, the revised transition is composed with the preceding and succeeding segments of the original behavior to obtain the revised behavior. Finally, the constraints introduced by the new component, represented by changes in the values of the parameters characterizing the old and new components (nitric-acid-pipe2 and sulfuric-acid-pipe2), are propagated forward through the newly composed behavior to obtain the revised internal behavior and function. Also, the schema for the function is revised by associating with it a pointer to the revised internal causal behavior.

Similarly, the structural difference *nitric acid=>sulfuric acid* between the structures of NAC and SAC is used to access the repair & evaluation plan for *substance-substitution*. The model-revision process for substance substitution further revises the (already revised) internal behaviors and functions in the model for NAC by replacing the old substance (low-acidity nitric acid) by the new substance (high-acidity sulfuric acid). This produces an SBF model for the SAC as shown in Figure 18 (only partial behaviors are shown). Then, the revised model is simulated (i.e., traced forward from the input state to the output state) to verify if it indeed results in the output behavior (i.e., the function) desired.

4.4. STORAGE OF NEW CASES

The final task in Kritik2's computational process is to store the new case in the case memory for later use. In order for a new design case to be recalled in later problem solving, Kritik2 needs to store it in the „right“ place. That is, it has to index the new design by its functions since Kritik2 solves *Function--to--Structure* mapping type of design tasks. Since the design case points to the SBF model of the design, the newly learned case-specific SBF model is also stored in Kritik2's memory.

4.4.1. Learning Indices to New Cases

In general, there are two different issues pertaining to the selection of functional indices for the new design case. First, if a new design is stored only along the substance properties specified in its function, case retrieval would not be able to make use of knowledge of other substance properties relevant to the design. Second, if the new design is indexed by all the properties of the substance in its functional specification, then case retrieval may result in a design based on a match with an unimportant property, which can make adaptation hard or even impossible. So, the issue becomes how to determine the substance properties that are *relevant* to the functioning of the design.

Kritik2 capitalizes on the knowledge of the causal behavior in the SBF models to address the above issues. In particular, it uses the behavioral requirements on the substance expressed under *under-condition-substance* to identify the substance properties relevant to the functioning of the design. These behavioral requirements of a substance specify that, in order for the transition to take place, the properties of the specified substance should satisfy certain conditions and hence are important to the design.

Kritik2's algorithm for selecting useful indices to a new case is shown in Figure 20. Given a new design case and the knowledge that functions are used to index the case, this method traverses through the causal behaviors in the SBF model of the design to identify substance properties on which the working of the design is predicated.

Since the SBF model can specify multiple behaviors, the outer loop (in step 1) in the algorithm analyzes each causal behavior in the model. The second loop is for analyzing the transitions within a causal behavior. If a substance property is a part of the causal context of a transition, then the algorithm adds it to the set of indexing features if it is a property of the containing substance in the functional specification, and to the set of alternative indexing features if it is a property of a contained substance. This results in using the properties of a contained substance (e.g. heat in Figure 18(b)) in the function as indexing features only when none of the properties of the containing substance (e.g. sulfuric acid in Figure 18(b)) are central to the design. Since the causal behaviors in Kritik2's SBF model are specified at different levels of detail, the algorithm searches the

space of behaviors in a breadth-first manner. If a higher level behavior does not lead to the identification of any useful substance properties, then the more detailed behavior, indicated by *by-behavior*, is added to the list of plausible sources of indexing features.

Input:

- Design case, *C*, that needs to be stored.
- Functional specification of the design. *F*.
- Type of indexing, *T*, that is, functional.
- One causal behavior (subset of model). *M*, corresponding to *F*.

Output: Exact vocabulary for indexing *C*. i.e., the set of useful features from *F*.

Procedure :

initialize

containing-substance-props *P* = get-containing-substance-properties(*F*);
 indices = alternative-indices = plausible-sources-of-indices = {};

while true do

1. **foreach** causal behavior *B* ∈ *M* **do**

• **foreach** transition *t* ∈ *B* **do**

- conditions-on-features = get-under-conditions-from-transition(*T*, *t*);
- indices = indices ∪ {*f* | feature *f* ∈ conditions-on-features ∧ *f* ∈ *P*};
- alternative-indices = alternative-indices ∪ {*f* | feature *f* ∈ conditions-on-features ∧ *f* ∉ *P*};
- if indices = *P* then exit(indices);
- if conditions-on-features = {} then plausible-sources-of-indices = plausible-sources-of-indices ∪ get-detailed-behavior(*t*);

end

end

2. **if** plausible-sources-of-indices = {} **then**

- **if** indices ≠ {} **then** exit(indices);
- **if** alternative-indices ≠ {} **then** exit(alternative-indices);
- indices = {*p* | *p* ∈ *P* ∧ input-state-value(*p*) ≠ output-state-value(*p*)};
 if indices ≠ {} **then** exit(indices);
- indices = {*p* | *p* ∈ get-contained-substance-properties(*F*) ∧
 input-state-value(*p*) ≠ output-state-value(*p*)};
 if indices ≠ {} **then** exit(indices);
- exit(*P*);

3. *M* = plausible-sources-of-indices;

4. plausible-sources-of-indices = {};

end

Figure 20: A model-based method to obtain functional indices for design cases

For the purpose of illustrating how Kritik2's model-based indexing method addresses these issues, let us now consider the task of identifying indices in our example, namely, the newly designed high-

acidity SAC (Figure 18), before storing it in memory. Although the case memory presently has the designs of acid coolers organized only along the dimension of property *acidity* (as shown in Figures 15 and 21(a)), the new design case may better be indexed along other dimensions also, so that it is more useful in later design problem solving. So, an important aspect of index learning task is to learn *new* indexing features. This results in the introduction of new dimensions of generalization/specialization of cases in the memory, and, thus, in a reorganization of the case memory.

Given the functional specification of high-acidity SAC (Figure 18(b)), and its causal behavior (Figure 18(c)), the above method results in acidity and state as the indexing features for storing this case in memory. This is because the annotation on *transition2*->3 specifies that the transition can occur only under certain condition relation to the state and acidity of the substance flowing through sulfuric-acid-pipe-2. The initial case memory (Figure 21(a)) did not have the property *state* as part of its indexing vocabulary. The SBF model however suggests that *state* is a useful index to the new case, and so Kritik2 indexes the new case by *state* also. The case memory after storing this design is shown in Figure 21(b).

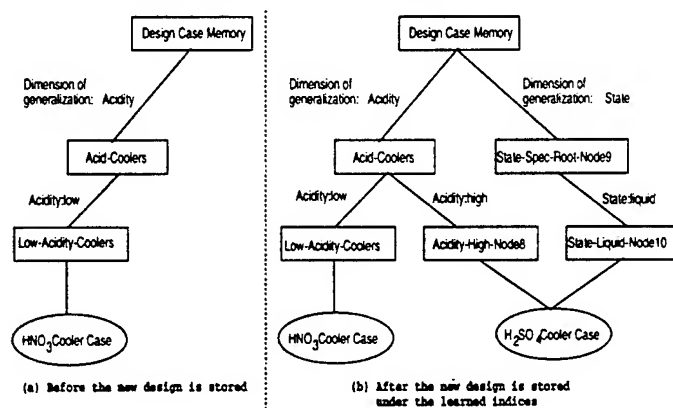


Figure 21: Snapshots of Kritik2's functionally organized case memory

Once the indexing features are selected, Kritik2 uses similarity-based learning to generalize the indices to the design cases. It uses the differences in parameters of a given property that constitute a type of functional difference between two designs to determine whether the two designs

belong to the same category or to different categories. For example, the design of high-acidity SAC is stored under the category of acidity-high-node8 that is different from that of low-acidity coolers (Figure 21 (b)). The level to which the indices are generalized depends on how similar are the corresponding parameters in the new and old cases in memory.

5. Evaluation of the Integrated Theory

In principle, Kritik2's integration of case-based and model-based reasoning can be evaluated in a number of dimensions such as (i) computational feasibility and efficacy; (ii) computational efficiency and scalability; (iii) generality in terms of domain independence; (iv) generality in terms of addressing different issues and tasks in case-based design; and (v) generality in terms of supporting case-based reasoning in the context of different tasks. We have built a growing family of systems for evaluating the integrated theory along many of these dimensions.

5.1. COMPUTATIONAL FEASIBILITY AND EFFICACY

Kritik (Goel, 1989, 1991a, 1991b, 1992) and Kritik2 (Bhatta and Goel, 1992; Stroulia and Goel 1992; Stroulia *et al.*, 1992) are working systems that integrate case-based and model-based reasoning for designing physical devices just as described here. These systems demonstrate that the integrated theory is computationally feasible.

They also demonstrate that the theory is quite effective in solving a range of problems in adaptive design. Kritik started with the designs of six devices stored in a flat case memory, along with their corresponding SBF models. Four of the initial designs were from the domain of electric circuits and two were from the domain of heat exchange devices. Kritik autonomously used the known designs and their models to solve four new design problems. Two of the new problems were in the domain of electric circuits including the one described here, and two in the domain of heat exchangers again including the one discussed above. For each of these four new problems, Kritik not merely solved the new problems presented to it, it also automatically acquired new designs and the associated SBF models for potential reuse. In one of these four experiments, Kritik reused its newly acquired design and SBF model for solving another problem.

Kritik2 is a newer, bigger and more complete implementation of Kritik. It incorporates both the theoretical and practical lessons we learned from Kritik. It presently contains some twenty five design cases and associated SBF models, and organizes them in a multi-dimensional hierarchy as outlined above. These cases are from five engineering domains including the domains of electric circuits and heat exchange devices in which Kritik operates. It also contains more repair plans than Kritik (such as the structure replication plan). Our experiments with Kritik2 indicate that the integrated theory is effective because the case-specific SBF models explicitly represent the internal behaviors of the known device, which specify not only the behavioral states and the state transitions, but also the functional role played by each structural component in these states and state transitions. In addition, it shows that SBF models also address the tasks of case retrieval and storage, and provide answers to the related issues of index learning and memory reorganization.

5.2. COMPUTATIONAL EFFICIENCY AND SCALABILITY

Our experiments with Kritik2 also indicate that the integrated theory is quite efficient due mainly to two reasons. First, the organization of the case memory and the functional indexing scheme enable the retrieval of cases relevant to the current problem, and second, the organization of the SBF models enable rapid localization of the search for „faults“ in the known case to a small portion of the SBF model. For instance, in the example of low-acidity nitric acid cooler, the substance-property-value difference plan needs to search only the internal causal behavior responsible for the function of cooling low-acidity nitric acid and can ignore all structural components that do not play any functional role in this behavior. This becomes possible because the functions in a SBF model act as indices to the causal behaviors responsible for them.

As we mentioned above, Kritik2 contains about twenty five design cases and device models. Therefore, while bigger than Kritik and many other case-based reasoning systems, it is a relatively small system. The scalability of the integrated theory thus remains an open issue. The main problem is in obtaining a large number of real designs, building SBF models for each of them, and entering them into Kritik2's case memory to bootstrap the design process. This is possible in principle but very

expensive in practice---the building of Kritik and Kritik2 has taken some six to seven years. A massive infusion of knowledge into these systems could be justified only after the feasibility of systems like Kritik2 had been demonstrated.

5.3. GENERALITY IN TERMS OF DOMAIN INDEPENDENCE

As mentioned above, Kritik designs simple electric circuits and heat exchange devices of the kind described here. Kritik2 operates in the additional domains of simple mechanical assemblies such as reaction wheels, electromagnetic devices, and electronic circuits such as operational amplifiers, and computer networks. This suggests that the integrated theory is not limited to any specific device domain. Our experiments with Kritik2, however, also indicate that the current version of the SBF language is inadequate for covering certain kinds of engineering devices. For example, the SBF language presently is well suited for representing devices whose function is to transform a behavioral state but needs additional primitives for representing devices whose function is to prevent the occurrence of a given behavioral state (e.g., a steam release valve in a steam chamber whose function is to prevent the pressure from becoming too high). Similarly, since the SBF language is based on a component-substance ontology, it presently does not provide primitives for drawing the needed inferences about fields such as the magnetic field (but see Goel, Stroulia, and Luk, 1994).

5.4. GENERALITY IN TERMS OF ADDRESSING DIFFERENT TASKS OF CASE-BASED REASONING

Kritik2 addresses all major tasks of case-based reasoning, not just the tasks of case retrieval and adaptation. This is important because each stage imposes constraints on the others. For example, the memory processes impose constraints on the kinds of problem solving that can be supported, problem-solving processes impose constraints on the kinds of learning that are needed, and the learning processes impose constraints on what is available in memory. We believe that simultaneously satisfying the constraints of memory, problem solving and learning is a critical aspect of the evaluation of any case-based system.

Note that Kritik2 uses the same representations of cases and their models for supporting the different tasks of case-based reasoning. Of course the inferences drawn, and therefore the role of the models, change from one stage of processing to another. The functional part of SBF models acts as index into the cases, and this enables Kritik2 to establish the similarity between the new problem and the cases in the retrieval stage. The language of the SBF models, that of the functional part in particular, provides a typology of functional differences between a known case and a new problem. The SBF models provide the functional and causal explanations of how the retrieved design works and this enables Kritik2 to infer the parts that need to be repaired in the adaptation stage. For instance, our experiments with Kritik2 indicate that SBF models are effective in solving the class of adaptation problems that can be characterized by the types of functional differences, such as, substance difference (i.e., the substances being transformed by the known design function and the new desired function are different), single substance-property-value difference, and multiple substance-property-value differences. The SBF models also enable the verification of the new design (i.e., behavior verification) by qualitative simulation in the evaluation stage. In addition, they provide the functional and causal explanations of how the candidate design works and this enables Kritik2 to learn new indices while storing the new case, and so on.

5.5. GENERALITY IN TERMS OF SUPPORTING CASE-BASED REASONING IN THE CONTEXT OF DIFFERENT TASKS

Recently we have started experimenting with the integration of case-based and model-based reasoning in different tasks and domains. In the Router project, for example, we are investigating the integration of case-based and model-based reasoning for navigation and planning (Goel *et al.*, 1994). While the results from this project are still preliminary, they indicate that the benefits of integrating case-based and model-based reasoning are not limited to design.

6. Related Research

Our work on the Kritik and Kritik2 systems is related to several lines of research in design and problem solving, case-based reasoning and learning, and qualitative models and model-based reasoning. The following discussion focuses on AI research on these topics.

Design and Problem Solving

AI research on design has led to the development, use and application of a number of knowledge-based problem-solving methods ranging from heuristic association (McDermott, 1982) to constraint satisfaction (Sussman and Steele, 1980) to plan instantiation (Mittal and Araya, 1992; Mittal *et al.*, 1986; Brown and Chandrasekaran, 1989), to reasoning from first principles (Williams, 1991). (Tong and Sriram (1992) provide a useful anthology of many important papers on knowledge-based design.) The adaptive approach of case-based reasoning is fundamentally different from these synthetic methods. Although the synthetic methods can and do play an important role in design adaptation, the adaptive approach views design in terms of evolution, in which new designs are created by modifying, perhaps combining, earlier designs.

AI research on case-based design has taken two distinct though related branches. In one branch, the emphasis has been on the theoretical development of case-based reasoning. This work has focused on developing and analyzing vocabularies and strategies for case representation, indexing, retrieval, adaptation, evaluation and storage. Kritik and Kritik2 are examples of this line of research; Hinrichs and Kolodner's (1991) Julia system is another prominent example. In the other, more popular branch, the emphasis has been on development and exploitation of case-based technology for aiding human designers in their tasks. This work has focused on developing and analyzing vocabularies and strategies for case representation, indexing, retrieval, and presentation. CADSYN and CASECAD (Maher *et al.*, 1995), CADET (Sycara *et al.*, 1991), CADRE (Hua and Faltings, 1993), and FABEL (Voss *et al.*, 1994) are some examples of this work. In our own work along this line, we have explored the use of case-based technology for aiding architects in the

preliminary design of office buildings (Pearce *et al.*, 1992) and for aiding software engineers in the task of interface design (Barber *et al.*, 1992).

Case-Based Reasoning and Learning

Several researchers (Ashley and Rissland, 1988; Hammond, 1989; Kolodner and Simpson, 1989) have developed computational models for case-based reasoning that posit different methods for adapting previous cases for solving new problems. The adaptation methods include *heuristic search* (Stallman and Sussman, 1977) and *heuristic association* (Hammond, 1989). The case-based method itself has been recursively used to adapt cases (Kolodner and Simpson, 1989; Goel *et al.*, 1994).

Research on case adaptation has generally followed the two main computational models of case-based reasoning, namely, transformational approach and derivational approach. Some case-based design systems, (e.g. Barletta and Mark, 1988; Dyer *et al.*, 1986; Hinrichs, 1992; Navinchandra, 1991; Maher and Zhao, 1987) generally follow the first computational model of case-based reasoning (Kolodner, 1993) in which the solutions to previous, similar problems are „tweaked“ to solve new problems. Some other case-based design systems, (e.g., Kambhampati, 1993; Mostow, 1989), closely follow the second computational model (Carbonell, 1983) in which the derivational trace of the problem solving in a previous design situation guides the adaptation process in the current situation.

Kritik and Kritik2 offer an alternative and complementary approach to case adaptation. A design case contains the specification of the design problem, the design solution, and a SBF model of how the design delivers the functions desired of the device. The SBF model gives rise to adaptation strategies and guides the adaptation process. This choice is due to both pragmatic and theoretical reasons. In real design situations, the design outcome, and especially the derivational record, often are not available, and are hard to encode when available. But a case-specific model of how the device works often is available, or can be reconstructed from the functional and structural specifications of its design, and Kritik2 provides a language for encoding it. From a theoretical perspective, Kritik2 provides an alternative account of how to automate case-based design in

which much of the design reasoning is in terms of the internal causal processes of physical devices.

Some other researchers too have explored model-based methods for case adaptation. For example, Koton (1988) has used causal domain models for comprehending diagnostic problems in internal medicine and retrieving appropriate diagnostic cases from memory, and Sycara and Navinchandra (1989) have proposed the use of causal domain models for elaborating engineering design problems, retrieving appropriate cases from memory, and adapting them. Simmons and Davis (1987) too have used causal domain models for debugging plans but only for testing modifications to a plan, not for generating the modifications. In contrast, Kritik2 system uses the model-based approach for all subtasks of case-based reasoning: case indexing and retrieval of similar cases from memory, generation of modifications to the retrieved design, evaluation and execution of the generated modifications, and index learning and storage of new cases in memory. In addition, Kritik2's SBF models are different from the causal models of Simmons and Davis, Koton, and Sycara and Navinchandra. The behavioral states and the state transitions in their models are grounded neither in the function nor in the structure of the system. In contrast, the SBF model explicitly relates the internal causal behaviors to both the function and the structure of a device, and thus constrains them both from the top and the bottom.

Interestingly, recent work on case-based design aiding (e.g., Hua and Faltings, 1993; Maher *et al.*, 95; Sycara *et al.*, 1991; Voss *et al.*, 1994) too has been moving towards the use of case-specific models to support the tasks of case retrieval and adaptation. Maher *et al.* use case-specific FBS models that are very similar to SBF models.

Qualitative Models and Model-Based Reasoning

Since our work uses a model-based approach, we will briefly compare it to some related research on device qualitative models and model-based reasoning. Research on naive physics and qualitative reasoning, (e.g., deKleer and Brown, 1984; Forbus, 1984; Kuipers, 1984), has focused on qualitative modeling and simulation of the physical world. The emphasis of this work has been on the content, representation and use of qualitative

models of physical systems, and the focus has been on the derivation of the system's behaviors at problem-solving time. In contrast, our work seeks to address the issues of organization, indexing, and acquisition of the qualitative models in addition to their content, representation and use. This has led us to the *structure-behavior-function* (SBF) models of physical devices.

Our memory-based view of device models is related to other work on memory-based approaches to comprehension, for example Minsky (1975) and especially Schank (1982). Schank (1982) describes Memory Organization Packets (MOPs) for representing and organizing certain kinds of information in a compiled form, for example, the goals of volitional actors, and the sequences of actions performed to achieve the goals. He also describes how MOPs can facilitate story interpretation and enable generalization and learning from past experiences. We adopt a similar view towards device models: SBF models organize functional, causal, and structural information underlying the functioning of devices. They facilitate tasks such as interpretation of design descriptions, design generation and evaluation, and learning from design experiences. They also provide the indexing vocabulary for organizing design cases in memory and enable automatic learning of the indices to new cases.

As mentioned earlier, SBF models are based on a component-substance ontology. They integrate and generalize two earlier device representations: the functional representation scheme (Sembugamoorthy and Chandrasekaran, 1986; Chandrasekaran, Goel and Iwasaki 1993) and the behavioral primitives of the consolidation method (Bylander and Chandrasekaran, 1985; Bylander, 1991). In addition, the SBF models are related to the commonsense algorithms of Rieger and Grinberg (1978). The representation of behavioral states and state transitions in our scheme is similar to their representations. The internal organization, the indexing scheme, and the typology of structural, causal, and behavioral relations in SBF models complement those used in the commonsense algorithms.

7. Conclusions

The Kritik and Kritik2 experiments lead us to the following four conclusions:

1. The computational process of case-based reasoning provides a good account of the variant and adaptive aspects of preliminary conceptual design of physical devices.
2. While case-based reasoning provides a high-level computational process, it also raises a number of issues pertaining to case content, representation, indexing, organization, retrieval, adaptation, evaluation and storage. The different issues and tasks in the case-based reasoning process impose constraints on one another. It is important to address all the different tasks and issues in order to develop a well-constrained theory of case-based design.
3. Structure-Behavior-Function device models capture a reasoner's comprehension of how a device works, i.e., how the structure of the device delivers its functions, how the internal behaviors of the device compose the functions of the structural components into the functions of the device as a whole. The SBF language is expressive enough to cover a large domain of physical devices and precise enough to support the inferences needed to address a large range of design tasks and subtasks.
4. The SBF theory provides a grounding for case-based design. In particular, the SBF theory provides an account of the case content, and vocabularies for case representation, indexing and organization. In addition, it provides strategies for the tasks for case retrieval, adaptation, evaluation and storage, including index learning, and memory reorganization.

Acknowledgements

This research has benefited from contributions by B. Chandrasekaran and S. Prabhakar. This work started when the first author was at the Ohio State University and after 1989 continued at Georgia Institute of Technology. At Georgia Tech, it has been supported by research grants from the National Science Foundation (grant C36-688), Office of Naval Research (contract N00014-92-J-1234), Northern Telecom, Georgia Tech Research Corporation, and a CER grant from NSF (grant CCR-86-19886), and equipment grants and donations from IBM, Symbolics, and NCR.

References

- Ashley, K., and Rissland, E. (1988) A case-based approach to modeling legal Expertise. *IEEE Expert*, 3(3):70--77.
- Barber, J., Bhatta, S., Goel, A., Jacobson, M., Pearce, M., Penberthy, L., Shankar, M., Simpson, R., and Stroulia, E. (1992) AskJef: Integrating Case-Based and Multimedia Technologies for Interface Design Advising. In *Proc. Second International Conference on Artificial Intelligence in Design*, Pittsburgh, 457--476.
- Barletta, R., and Mark, W. (1988) Breaking cases into pieces. In *Proc. of the AAAI Workshop on Case-Based Reasoning*, 12--17.
- Bhatta, S. and Goel, A. (1992) Use of mental models for constraining index learning in experience-based design. In *Proc. of the AAAI workshop on Constraining Learning with Prior Knowledge*, 1--10, San Jose, CA.
- Bhatta, S. and Goel, A. (1995) Model-Based Indexing and Index Learning in Analogical Design. In *Proc. Seventeenth Annual Conference of the Cognitive Science Society*, Pittsburgh, July 22--25, NJ, Hillsdale: Erlbaum.
- Brown, D.C., and Chandrasekaran, B. (1989) *Design Problem Solving: Knowledge Structures and Control Strategies*. Pitman, London, UK.
- Bylander, T., and Chandrasekaran, B. (1985) Understanding behavior using consolidation. In *Proc. of the Ninth International Joint Conference on Artificial Intelligence*, 450--454.
- Bylander, T. (1991) A theory of consolidation for reasoning about devices. *International Journal of Man-Machine Studies*, 35(4):467--489.
- Carbonell, J. (1983) Learning by analogy: Formulating and generalizing plans from past experience. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Tioga, Palo Alto, CA.
- Chandrasekaran, B., Goel, A. and Iwasaki, Y. (1993) Functional Representation As Design Rationale, *IEEE Computer*, January, 48--56.
- Dyer, M., Flowers, M., and Hodges, J. (1986) Edison: An engineering design system operating naively. In *Proc. of the First International Conference on AI Applications in Engineering*.
- deKleer, J. and Brown, J. (1984) A qualitative physics based on confluences. *Artificial Intelligence*, 24:7--83.
- Forbus, K. (1984) Qualitative process theory. *Artificial Intelligence*, 24:85--168.
- Goel, A. (1989) *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving*. PhD thesis, The Ohio

State University, Department of Computer and Information Science,
Columbus, Ohio.

- Goel, A. and Chandrasekaran, B. (1989a) Use of Device Models in Adaptation of Design Cases. In *Proc. Second DARPA Case-Based Reasoning Workshop*, Pensacola, Florida, 100-109, Los Altos, CA: Morgan Kaufmann.
- Goel, A. and Chandrasekaran, B. (1989b) Functional Representation of Designs and Redesign Problem Solving, In *Proc. of IJCAI-89*, 1388-1394.
- Goel, A. (1991a) A model-based approach to case adaptation. In *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society*, 143--148, Chicago.
- Goel, A., (1991b) Model Revision: A Theory of Incremental Model Learning, In *Proc. Eighth International Workshop on Machine Learning*, 605--609
- Goel, A. and Prabhakar, S. (1991) A Control Architecture for Model-Based Redesign Problem Solving. In *Proc. IJCAI-1991 Workshop on AI in Design*, Sydney, Australia.
- Goel, A. (1992) Representation of design functions in experience-based design. In D. Brown, M. Waldron, and H. Yoshikawa, (editors), *Intelligent Computer Aided Design*, 283--308. North-Holland, Amsterdam, Netherlands.
- Goel, A. and Chandrasekaran, B. (1992) Case-Based Design: A Task Analysis. In *Artificial Intelligence Approaches to Engineering Design, Volume II: Innovative Design*, C. Tong and D. Sriram (editors), . 165--184, San Diego: Academic Press.
- Goel, A., Stroulia, E., and Luk, K.Y. (1994) Functional Reasoning about Devices with Fields and Cycles. In *Proc. AAAI-94 Workshop on Representation and Reasoning about Function*, Seattle, Washington, 48--55.
- Goel, A., Ali, K., Donnellan, M., Gomez, A., and Callantine, T. (1994) Multistrategy Adaptive Navigational Path Planning. *IEEE Expert*, 9(6):57--65.
- Hammond, K. (1989) *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, MA.
- Hinrichs, T.R., and Kolodner, J.L. (1991), The Roles of Adaptation in Case-Based Design, In *Proc. of the Ninth National Conference on AI*, MIT Press.
- Hinrichs, T.R. (1992) *Problem Solving in Open Worlds: A Case Study in Design*. Erlbaum.
- Hua, K. and Faltings, B. (1993) Exploring Case-Based Building Design - CADRE. In *AI(EDAM)*, 7(2):135--143.
- Kambhampati, S. (1993) Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence*.

- Kolodner, J.L., and Simpson, R.L. (1989) The mediator: Analysis of an early case-based problem solver. *Cognitive Science*, 13(4):507--549.
- Kolodner, J.L. (1993) *Case-Based Reasoning*. Morgan-Kaufmann, San Mateo, CA.
- Koton, P. (1988) Integrating case-based and causal reasoning. In *Proc. of the Tenth Annual Conference of the Cognitive Science Society*, Northvale, NJ, Erlbaum.
- Kuipers, B. (1984) Commonsense reasoning about causality. *Artificial Intelligence*, 24:169--203.
- Maher, M.L. and Zhao, F. (1987) Using experiences to plan the synthesis of new designs. In J.S. Gero, editor, *Expert System in Computer-Aided Design*, 349--369. North-Holland, Amsterdam.
- Maher, M.L., Balachandran, M.B., Zhang, D. (1995) *Case-Based Reasoning in Design*, Hillsdale, NJ: Erlbaum.
- McDermott, J. (1992) R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19:39--88.
- Minsky, M. (1975) A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, 211--277. McGraw-Hill, New York.
- Mittal, S., Dym, C., and Morjaria, M. (1986) Pride: An expert system for the design of paper handling systems. *Computer*, 19(7):102--114.
- Mittal, S., and Araya, A. (1992) A knowledge-based framework for design. In C. Tong and D. Sriram, editors, *Artificial Intelligence in Engineering Design, Vol. I*. Academic Press, San Diego, CA.
- Mostow, J. (1989) Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40:119--184.
- Navinchandra, D. (1991) *Exploration and Innovation in Design: Towards a Computational Model*. Springer-Verlag, New York.
- Pearce, M., Goel, A., Kolodner, J.L., Zimring, C., Sentosa, L., and Billington, R. (1992) Case-Based Decision Support: A Case Study in Architectural Design. *IEEE Expert*, 7(5):14-20.
- Pittges, J., Eiselt, K., Goel, A., Gomez, A., Mahesh, K., and Peterson, J. (1993) Ka: Integrating natural language processing and problem solving. In *Proc. of the Fifteenth Annual Conference of the Cognitive Science Society*, 818--823, Boulder, CO.
- Rieger, C., and Grinberg, M. (1978) A system for cause-effect representation and simulation for computer-aided design. In J. Latombe, editor, *Artificial Intelligence and Pattern Recognition in Computer-Aided Design*, 299--334. North-Holland, Amsterdam, Netherlands.

- Riesbeck, C., and Schank, R. (1989) *Inside Case-Based Reasoning*. Hillsdale, NJ: Erlbaum.
- Schank, R. (1982) *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, Cambridge.
- Sembugamoorthy, V., and Chandrasekaran, B. (1986) Functional representation of devices and compilation of diagnostic problem-solving systems. In J.Kolodner and C.Riesbeck, editors, *Experience, Memory and Reasoning*, 47--73. Lawrence Erlbaum, Hillsdale, NJ.
- Simmons, R., and Davis, R. (1987) Generate, test and debug: Combining associational rules and causal models. In *Proc. of the IJCAI-87*, San Mateo, CA, Morgan Kaufmann.
- Stallman, R., and Sussman, G. (1977) Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135--196.
- Stroulia, E. and Goel, A. (1992) Generic Teleological Mechanisms and their Use in Case Adaptation, In *Proc. of the Fourteenth Annual Conference of the Cognitive Science Society*, 319--324.
- Stroulia, E., Shankar, M., Goel, A., and Penberthy, L. (1992) A model-based approach to blame-assignment in design. In J.S. Gero, editor, *Proc. of the Second International Conference on AI in Design*, 519--537, Pittsburgh.
- Sussman, G., and Steele, G. (1980) Constraints: A language for expressing almost-hierarchical descriptions. *Artificial Intelligence*, 14:1--39.
- Sycara, K., and Navinchandra, D. (1989) Integrating case-based reasoning and qualitative reasoning in engineering design. In J.S.Gero, editor, *Artificial Intelligence in Engineering*.
- Sycara, K., Navinchandra, D., Guttal, R., Koning, J., and Narasimhan, S. (1991) CADET: A Case-Based Synthesis Tool for Engineering Design *International Journal of Expert Systems*, 4(2):157-188.
- Tong, V., and Sriram, D. (editors) (1992) *Artificial Intelligence in Design*, Volumes I, II, and III, Boston: Academic Press.
- Voss, A., Coulon, C-H., Grather, W., Linowski, B., Schaaf, J., Barstsch-Sporl, B., Borner, K., Tammer, E., Durscke, H., and Knauff, M. (1994) Retrieval of Similar Layouts - About a Very Hybrid Approach in FABEL. In *Proc. of Fourth International Conference on AI in Design*, Lausanne, 625--640.
- Williams, B. (1991) Interaction-based design: Constructing novel devices from first principles. In *Procs. of the IFIP International CAD Conference*.

Situating natural language understanding within experience-based design

JUSTIN PETERSON, KAVI MAHESH AND ASHOK GOEL

College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

(Received 21 June 1994 and accepted in revised form 25 September 1994)

Building useful systems with an ability to understand "real" natural language input has long been an elusive goal for Artificial Intelligence. Well-known problems such as ambiguity, indirectness, and incompleteness of natural language inputs have thwarted efforts to build natural language interfaces to intelligent systems. In this article, we report on our work on a model of understanding natural language design specifications of physical devices such as simple electrical circuits. Our system, called KA, solves the classical problems of ambiguity, incompleteness and indirectness by exploiting the knowledge and problem-solving processes in the situation of designing simple physical devices. In addition, KA acquires its knowledge structures (apart from a basic ontology of devices) from the results of its problem-solving processes. Thus, KA can be bootstrapped to understand design specifications and user feedback about new devices using the knowledge structures it acquired from similar devices designed previously.

In this paper, we report on three investigations in the KA project. Our first investigation demonstrates that KA can resolve ambiguities in design specifications as well as infer unarticulated requirements using the ontology, the knowledge structures, and the problem-solving processes provided by its design situation. The second investigation shows that KA's problem-solving capabilities help ascertain the relevance of indirect design specifications, and identify unspecified relations between detailed requirements. The third investigation demonstrates the extensibility of KA's theory of natural language understanding by showing that KA can interpret user feedback as well as design requirements. Our results demonstrate that situating language understanding in problem solving, such as device design in KA, provides effective solutions to unresolved problems in natural language processing.

1. Introduction and overview

It has long been recognized that language understanding requires abilities far beyond what pure linguistic knowledge permits (Kintsch, 1974; Charniak & McDermott 1985; Grishman, 1986; Rich & Knight, 1991; Winston, 1992). In the past, two approaches have been pursued to endow natural language understanding systems with such abilities as inference and problem solving. One approach has been to endow systems with a variety of domain and world knowledge as well as a range of inference, explanation and reasoning capabilities (e.g. BORIS: Lehnert, Dyer, Johnson, Yang & Harley, 1983). All of the system's capabilities, in this approach, are solely employed in the service of "Understanding" natural language. Unfortunately, this approach to language understanding has not been particularly successful, since the linguistic methods typically used for "Understanding" do not work well with the various types of non-linguistic knowledge that are required to understand

language. Moreover, the specialized types of knowledge that these systems require for understanding a natural language text (Lehnert *et al.*, 1983) are not readily available, nor has it been demonstrated that they can be acquired by established methods. As a result, this approach has only resulted in prototype language understanding systems that show little promise of scalability or bootstrapping.

Another approach has been to make some other cognitive task (such as robot planning or expert decision making) the main task and add a natural language "Front-end" to the system. The front-end works in service of the rest of the system and has limited abilities to translate natural language inputs into a conceptual representation that is comprehensible to the rest of the system (e.g. Hendrix, Sacerdoti, Sagalowicz & Slocum, 1978; Simon & Hayes, 1979). Such natural language front-ends are unable to solve many linguistic problems because they neither possess the requisite non-linguistic knowledge nor get any useful feedback from the other task. The linguistic problems remain hidden in the intermediate representation and are not resolved satisfactorily by the rest of the system with its non-linguistic methods and knowledge.

Our work on natural language understanding takes a third approach, a more modular one, in which language understanding and problem solving interact by communicating the results of their decision-making with each other. Language understanding uses the results of problem-solving operations to resolve linguistic problems such as ambiguities. Problem solving in turn uses the decisions made by language understanding to direct the course of its own problem decomposition and problem-solving process. A major difference between this approach to language understanding and previous approaches is that the language understander need not possess either the knowledge or the abilities to solve problems in reasoning. Nor does problem solving need to know how to solve linguistic problems left unsolved by a natural language front-end. All that the two need is to solve their own problems partially, be able to communicate their decisions and results with each other, and cooperate in an integrated architecture to arrive at a negotiated solution to the overall problem.

In this approach to building natural language understanding systems, we are not simply adding additional types of knowledge to a linguistic processor in the hope of making language understanding a feasible task. Nor are we passing on linguistic problems to a non-linguistic reasoning system in the hope that the reasoning system will somehow solve the problems in the natural language input. Instead, we are "Situating" natural language understanding in another task to make it more achievable—to exploit the knowledge and the reasoning processes running in the situation of another task to solve classical problems in natural language processing.

We have chosen to investigate the "situatedness" of natural language understanding within the design of simple physical devices such as electrical circuits. Design, like natural language understanding, is an oft-studied problem in AI, and many types of knowledge structures and reasoning methods have been developed for automating design, both in our own research and that of many others. More importantly, the knowledge of physical devices and their design that we are proposing to use in language understanding has been shown to be obtainable from prior problem-solving experiences (Goel, 1989, 1991a, b). Thus, the choice of physical device design for situating natural language understanding has a real

promise of scalability and bootstrapping if the knowledge that can be acquired in the problem-solving process can be used to solve classical problems in natural language such as ambiguity.

Taking this approach, we have built a natural language understanding system, called KA, for (1) understanding device specifications written in natural language (English) in the context of designing new devices and (2) understanding user feedback in natural language in the context of device design. KA embodies an integrated model-based and case-based approach to design problem solving that we have been developing for many years now (Goel, 1989, 1992; Goel & Chandrasekaran, 1989, 1992). KA uses the same approach to address well-known problems of natural language understanding such as resolving ambiguity, interpreting indirect statements and inferring unspecified information. We have conducted several investigations to demonstrate that KA's design situation provides viable solutions to the problems of ambiguity, indirectness and omission.

In this article, we describe three investigations in solving natural language understanding problems with KA. Our first investigation demonstrates that KA's models of physical devices and its reasoning for their design helps resolve ambiguities in design specifications as well as infer unarticulated requirements. Briefly, our approach was to construct an initial, tentative interpretation of the design specification using KA's language processing capabilities, locate a similar design in KA's case memory using model-based retrieval, and then use the retrieved design to countermand erroneous decisions made in the resolution of ambiguities. These retrieved designs were also used to augment the interpretation of requirements with those that were not articulated in the natural language specification. Our results in this first investigation indicate that, among other benefits, models of physical devices and the ability to reason about the function of devices aid ambiguity resolution in two specific ways. First, the ontology of physical devices employed in KA grounds the semantic representations of language processing, ensuring that decisions about the consistency of interpretations are made in accordance with the ontology of device design. Second, by allowing previous problem-solving experiences to be factored into linguistic decision-making, interpretations that are most compatible with past experience are produced.

The second investigation demonstrates that KA's problem-solving capabilities help ascertain the relevance of indirect design specifications, and identify unspecified relations between detailed requirements. Our approach to these problems relied extensively on KA's memory of design cases, case-specific models of devices, and model-based methods for design adaptation. Our results indicate that a memory of design cases and device models as well as the ability to adapt these descriptions in accordance with a deep understanding of the structure, function and behavior of devices provides considerable leverage when dealing with indirect and ill-specified English descriptions of design problems. Using device descriptions in memory as baseline interpretations and the information extracted from the text as constraints on interpretation, model-based adaptation proves to be an effective means of producing both an interpretation of the text and a successful design solution. This result, along with those of the first investigation, demonstrates that situating natural language understanding in design problem solving provides tractable solutions to problems in understanding natural language specifications of design problems.

The third investigation demonstrates that KA's approach to situated natural language understanding can be extended to other, related situations. Natural language texts are used to achieve a variety of communication goals at different stages in the design process. For example, at later stages in the design, customers use English texts to communicate feedback to designers which the designers must understand in order to redesign the device. In the third and most recent investigation, we chose a problem in the design of a reaction wheel assembly for the Hubble space telescope and examined KA's ability to understand user feedback. We demonstrated that KA was in fact able to understand such user feedback and use the information it could extract from the feedback to redesign the reaction wheel assembly. This cost effective redesign was made possible by KA's repertoire of plans for incremental redesign to correct the malfunction and its ability to precisely identify the part of the design that is to blame for the malfunction. This investigation demonstrated that some of the same kinds of knowledge and methods that are used in understanding initial design requirements also enable the integrated system to understand natural language feedback from the customer for iterative redesign to meet customer requirements.

The organization of the rest of this article is as follows. First, we provide a brief description of the design situation, the ontology of physical devices, and the problem-solving methods used in KA's model of designing physical devices. Next, we show how classical problems in natural language understanding get redefined when the understanding is situated in the design task. The section after this describes the solution to these problems in terms of the architecture of the KA system, detailing the different components that make up KA. Following this description, we present the three investigations that substantiate our claim that situated understanding provides viable solutions to problems in natural language understanding. This presentation will include sample texts used in our work, an analysis of their difficulties, and a demonstration of KA's capabilities. Then, we describe the strengths and limitations of our work and compare it to that of other research projects. We conclude by articulating the contributions of our research. Throughout the paper, we focus on language processing. The reader may refer to our earlier papers that describe memory and problem solving in greater detail.

2. The design situation

The task in which we are situating natural language understanding is the design of simple physical devices such as electrical circuits and computer networks (Peterson, Mahesh, Goel & Eiselt, 1994). Our work evolves from previous work on the Kritik project which used past design cases and associated device models for creating new device designs (Goel, 1989). Such prior experiences in design are stored in KA's memory in the form of cases and case-specific models (Goel, 1989, 1991a, b, 1992). The memory uses a representation called the *SBF language* (Sembugamoorthy & Chandrasekaran, 1986; Goel, 1989, 1992; Chandrasekaran, Goel & Iwasaki, 1993), based on a component-substance ontology of the domain of physical devices (Bylander & Chandrasekaran, 1985; Goel, 1989; Bylander, 1991). In the SBF language, a device is represented in terms of its desired function, its internal structure (components and connections between them), and internal behaviors of its

components. These behaviors are articulated in terms of the various *substances*[†] contained in the components, the states of the substances, and the flow of substances between components.

Given a functional specification of a device to be designed, previous designs of functionally similar devices are retrieved from the cases and models in memory. The models in the previous cases are adapted to the present problem using model-based methods (Goel, 1991a, b). KA's design capabilities extend, however, far beyond mere case-based adaptation of prior designs. It can diagnose faulty designs, identify the components at error, and redesign the device by applying various design repair plans that it knows about, such as component replacement and component cascading (Goel, 1991a, b; Stroulia & Goel, 1992; Stroulia, Shankar, Goel & Penberthy, 1992). Interestingly, it can also acquire new experiences and store away the new cases and models in its memory by learning appropriate indices to the cases and models (Goel, 1991a, b; Bhatta & Goel, 1992, 1993).

2.1. THE SBF LANGUAGE

Since our investigations will be using examples described in the SBF language, we make a brief digression in this section to describe the salient terms in the language. Models of physical devices are represented in terms of their structure, behavior and function (SBF). These models are based on a *component-substance ontology*. In this ontology, the structure of a device is constituted of its *components* and *substances*. Substances have *locations* in reference to the various components of the device. They also have *behavioral properties*, such as *voltage of electricity*, and corresponding *parameters*, such as *1.5 volts*, *3 volts*, and so on. This ontology gives rise to the SBF language.

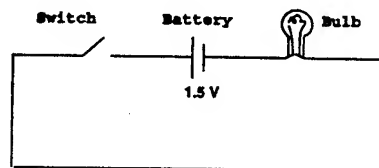
2.1.1. Structure

The structure of a design is expressed in terms of its constituent components and substances and the interactions between them. Figure 1(a) shows the structure of a 1.5-volt electric circuit (EC1.5) schematically.

2.1.2. Function

A function is represented as a schema that specifies the behavioral state the function takes as input, and the behavioral state it gives as output. Figure 1(b) shows the function "Product Light" of EC1.5. Both the input state and the output state are represented as *substance schemas*. The input state specifies that electricity at location Battery in the topography of the device [Figure 1(a)] has the property voltage and the corresponding parameter 1.5 volts. The output state specifies the property intensity and the corresponding parameter 6 lumens of a different substance, light, at location Bulb. A third aspect of a functional specification is

[†] The term substance as used here not only includes material substances such as air and water, but also abstract substances and forms of energy such as heat, electricity, information, force, and so on (Bylander, 1991).



(a) 1.5-volt Electric Circuit (EC1.5)

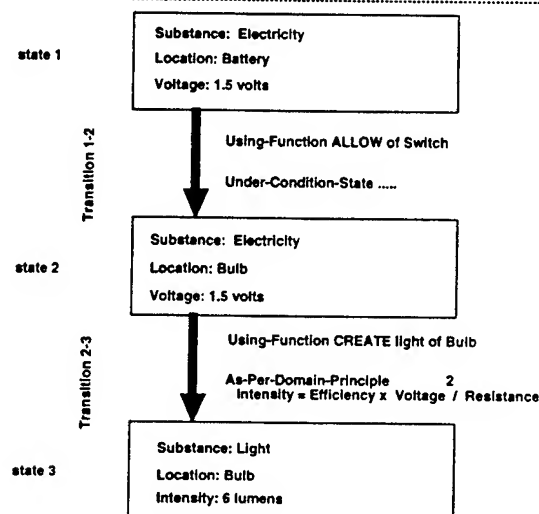
Input: Substance: Electricity
Location: Battery
Voltage: 1.5 volts

Output: Substance: Light
Location: Bulb
Intensity: 6 lumens

Stimulus: Substance: Force
Location: Switch

By-Behavior: "Produce Light"

(b) Function of Circuit EC1.5



(c) Behavior "Produce Light" of EC1.5

FIGURE 1. SBF model of a 1.5-volt electric circuit (EC1.5).

the *stimulus* which initiates the behaviours of the device. For example, the force on the switch is the stimulus to the electrical circuit in Figure 1. In addition, the slot *by-behavior* acts as an index into the causal behavior that achieves the function of producing light.

2.1.3. Behavior

The internal causal behaviors of a device are viewed as sequences of *state transitions* between *behavioral states*. The annotations on the state transitions express the *causal*, *structural*, and *functional context* in which the transformation of state

variables, such as substance location, properties and parameters, occur. Figure 1(c) shows the causal behavior that explains how electricity in Battery is transformed into light in Bulb. *State*₂ is the preceding state of *transition*₂₋₃ and *state*₃ is its succeeding state. *State*₁ describes the state of electricity at location Battery and *state*₂ at location Bulb. *State*₃ however describes the state of light at location Bulb. The annotation USING-FUNCTION in *transition*₂₋₃ indicates that the transition occurs due to the primitive function "create light" of Bulb.

3. Situating natural language understanding

We now return to the task of natural language understanding and show how situating it in the design situation redefines classical problems such as ambiguity, indirectness and incompleteness. We also show how the design situation suggests workable solutions to these linguistic problems. Real world tasks such as designing physical devices from written requirements specifications provide a context which refocuses many of the linguistic problems that have been central to the field, allowing us to consider novel solutions to time-worn yet unresolved problems. It also allows us to consider problems particular to texts that are currently hampering efforts to develop robust text understanding systems.

In taking this situated approach to language understanding, we have found that linguistic problems and the problems of large texts that are inherent to written design requirements actually become problems which require reasoning about the design of the device. Requirements specifications are notoriously confusing and incomplete, providing poor articulations of the design requirements. In the KA project, we have encountered the following problems in requirements specifications:

- **Ambiguity.** The natural language surface form has multiple mappings into a conceptual representation of the device.
- **Incompleteness.** The natural language surface form fails to articulate a design requirement.
- **Indirectness.** The natural language surface form indirectly refers to a design requirement.
- **Underspecification.** The natural language surface form does not indicate certain relationships between design requirements.

Because research in natural language understanding has so decidedly separated the problems of linguistic analysis and sentence understanding from the other problems that must be resolved in the meaningful interpretation of texts, the linguistic solutions that have been proposed in most text understanding systems have been severely limited. For quite some time, the conventional wisdom has been that problems in natural language understanding are best addressed by constraint-based methods that employ a knowledge of natural language's distributional structure and rules of combination (Chomsky, 1957). In KA, we take a different approach, where the design situation provides the knowledge and results of applying its reasoning methods that are then used to solve the above problems in language understanding.

Below, we take each of these problems individually, identifying the conditions under which they occur and elaborating on their consequences.

In general, the mapping of language form to design requirements is *ambiguous*. For example, words as seemingly clear as "input" have multiple mappings into an SBF representation of function. The "input" to a device may refer to either an external stimulus (e.g. a force on a switch that initiates some causal behavior) or some entity that is transformed by the device (e.g. a substance like electricity consumed by an electrical device such as a light bulb). Because such ambiguities crop up often, requirement specifications written in natural language frequently specify several devices, rather than a single, unambiguous device. Leaving these ambiguities unresolved or failing to resolve them correctly cause a system to waste its resources pursuing a number of fruitless design efforts.

In general, requirement specifications written in natural language are also *incomplete*. For example, although electrical devices require a source of power, design problems can fail to mention how this power is to be derived. The device could use batteries, plug into an electrical outlet, or resort to some other electrical power source. It is critical that the system infer the appropriate design requirement because each of these designs would entail different structures and would be operable under differing conditions.

In general, requirements specifications state the design requirements *indirectly*. They refer to aspects of the device that are only distantly related to its principal features. For example, specifications for computing devices often identify principal components, specify the inputs and outputs of these components, and delineate their connectivity, but fail to define the big picture, viz. the general functions of the device. The writers of such requirements specifications can usually point out specific statements about the inputs and the outputs of the components, for example, that indicate the general functions of the device, but in no way are these requirements indicated in the natural language surface form. The system must be able to use the indirect statements given to infer the design requirements because, failing to do so, it would be unable to pursue a design solution.

In general, natural language specifications are *underspecified*. They identify detailed device requirements without articulating how these requirements relate to one another. For example, although baud rate, size of an information packet and frequency of transmission have a well-defined relationship to one another in a computer network, requirements specifications for computing devices rarely, if ever, mention this relationship. A superficial analysis of the natural language surface form would produce three separate requirements (one for the baud rate, one for information pack size and one for the frequency of transmission), entailing an extremely inefficient problem decomposition. If the system is to pursue designs efficiently, it must combine these disparate requirements into a coherent specification of the design.

In order to map requirement specifications to useful functional descriptions in the SBF language, KA must effectively resolve ambiguity, fill in missing details, identify the relevance of indirect statements, and combine related information. To do so efficiently, KA uses memory, comprehension and problem-solving processes in addition to purely language processes. In this way, the design situation in KA provides a robust context in which effective comprehension of natural language becomes feasible.

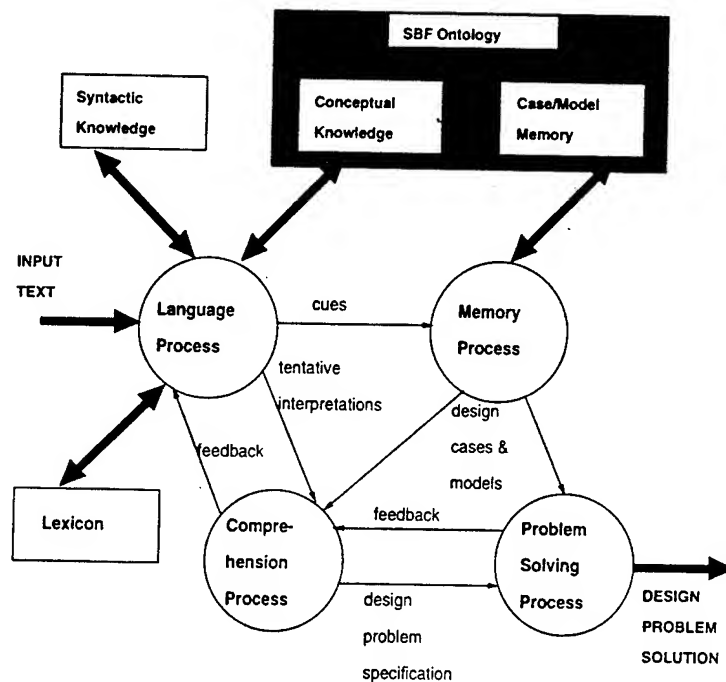


FIGURE 2. KA system architecture.

4. The KA architecture

KA is a *case/model-based text interpretation and design problem-solving system* which accepts a requirements specification written in English and produces a design expressed as a structure-behavior-function (SBF) model, which meets the design requirements. The functional architecture for KA is illustrated in Figure 2. It consists of several knowledge sources containing syntactic, conceptual and episodic knowledge, and employs memory, comprehension and problem-solving processes in addition to a language process.

4.1. KNOWLEDGE SOURCES IN KA

The component processes in KA use different knowledge sources to bring about the capabilities of the system. The knowledge sources are,

- The **Lexicon**, which contains knowledge of the words in the language. It provides such information about words as their grammatical category, other linguistic markers such as number and person, and their conceptual meaning.
- **Syntactic knowledge**, which is knowledge of the grammar of the natural language. This is used by the language process to break up the input sentences into grammatical units such as phrases and clauses.
- **Conceptual knowledge**, which in KA is the knowledge of substances, components, their properties, states and functions. Conceptual knowledge is essentially the content of the domain of physical devices and is expressed in the SBF ontology.

- **Case/Model Memory**, which is the episodic memory of past design cases and case-specific structure-behavior-function (SBF) device models. Like conceptual knowledge, this knowledge is represented in the SBF ontology and indexed by items in the ontology such as functions and properties of substances and components.

4.2. PROCESS IN KA

Below, we describe in detail each of the processes in KA that utilize the knowledge sources to accomplish the task of KA.

4.2.1. *Language*

If the KA system is to effectively comprehend natural language texts, it must be able to resolve the different types of ambiguities (e.g. lexical and structural ambiguities) that arise in written texts. The language process in KA uses an early-commitment processing strategy with robust error-recovery to resolve word sense ambiguities (Eiselt, 1987, 1989). This mechanism has proved itself to be quite effective. Its early-commitment strategy provides the system with the ability to pursue a tentative interpretation of the discourse. This allows the system to discover the entailments of this line of interpretation, bringing other processes on-line early in the course of language understanding. In situations where the early decision is incorrect, the error-recovery mechanisms may use feedback from the comprehension (or problem-solving) process to reactivate a previously retained alternative interpretation. The output of the language processor serves both as a set of cues for the memory process and as a tentative interpretation for the comprehension process.

The language process consists of two components: a parser which produces syntactic structures and a semantic network that produces conceptual interpretations. Consistent with the early-commitment processing strategy, the semantic network resolves word-sense ambiguities by considering processing choices in parallel, selecting the alternative that is consistent with the current context, and deactivating but retaining the unchosen alternatives for as long as space and time resources permit. If some later context proves the initial decision to be incorrect, retained alternatives are reactivated without reaccessing the lexicon or reprocessing the text (Eiselt, 1989).

4.2.2. *Memory*

The memory process retrieves and stores design knowledge from an episodic memory that contains both design cases and case-specific device models. Design cases specify a design problem encountered by KA in the past and its corresponding solution. A case-specific SBF model of a known device specifies the causal behaviors that explain how the structure of the device produces the device functions. In order to ensure effective retrieval, the cases are indexed by the functional specification of the stored design and the SBF models are indexed by the cases.

4.2.3. *Comprehension*

The comprehension process provides feedback to the language process based on the retrieved cases and associated SBF models. It also generates new SBF device models by retrieving and adapting previously encountered design cases and their SBF models. Based on information provided by the language process, the retrieved

design and its model are adapted using generic design repair plans. The comprehension process selects these modification plans by using the differences between the functions of the new device and the functions of the retrieved design as an index.

4.2.4. Problem solving

The problem-solving process performs *function-to-structure* design tasks. It accepts a *functional* specification of the desired design as input and produces a *structural* specification that realizes the specified function as output. Both the specification of function and the specification of structure are articulated in terms of the SBF language.

The problem-solving process begins its task by soliciting the memory process for a case that most closely matches the functional specification of the desired design. The memory process returns an SBF model which problem solving uses to adapt the design's structure so as to meet the given functional specification. Model-based diagnosis is used to identify the modifications needed to the retrieved design, and repair plans are used to perform these modifications to the design's structure. Once the structural modifications are completed, this new design is verified by a qualitative simulation of its SBF model and produced as a solution to the design problem.

It may be noted from Figure 2 that the language process is neither a front-end to the problem-solving or comprehension processes, nor does it perform the entire understanding task by itself. What we have in KA is a highly interactive architecture in which language, memory, problem-solving and comprehension processes, each with its own sources of knowledge and its own capabilities, cooperate with each other, feeding back one's results and decisions to others, in order to arrive at an iterative solution to the overall problem of designing physical devices given their natural language specifications.

5. Investigation 1

In our first investigation, we examined whether KA's design situation and ability to reason about the design of physical devices could help resolve ambiguities in design specifications as well as infer unarticulated requirements. In this investigation, we sought to take advantage of the SBF ontology and KA's memory of past design cases and associated case-specific SBF models. Our results were very positive. They indicated the following benefits:

- Grounding the language process' conceptual knowledge in the SBF representation guarantees that decisions about the consistency of conceptual interpretations are made on the basis of their consistency as designs represented in the SBF ontology.
- Providing feedback to the language process in the form of past design cases, represented in the SBF language, ensures that the conceptual interpretations are compatible with past design experience and allows unarticulated design requirements to be inferred from previous design problems.

In this section, we discuss how each of these benefits was accrued in the

Consider a flashlight circuit. The function of the circuit is to produce light. The input is a small force on the switch. The output is light of eighteen lumens intensity and blue color.

FIGURE 3. Text 1—sample requirements specification.

implementation of KA. First, we present a sample text that is both ambiguous and incomplete along with its desired mapping into the SBF representation. Then, we demonstrate how KA performs the mapping from the natural language description to the functional specification of the desired design.

5.1. THE TASK

Figure 3 shows a sample *input* specification and Figure 4 shows its corresponding mapping, an SBF description of the desired design.

This simple example illustrates two general problems of natural language understanding that KA must solve. First, the requirement specification in Figure 3 is ambiguous. It states that the "input" to the device is a "small force on the switch", but in this domain, "input" can refer to one of two things, either an external stimulus (i.e. a force on a switch that initiates some causal behavior) or some entity that is transformed by the device (e.g. a substance, such as electricity, consumed by an electrical device). To produce the correct interpretation in Figure 4, KA must determine that in this instance "input" refers to an external stimulus by successfully resolving this lexical ambiguity.

Second, the requirement specification in Figure 3 is incomplete. There is no mention of how the device is to be powered. The description could be specifying a design which uses batteries or one which plugs into an outlet. In order to produce the functional specification in Figure 4, KA must infer that the design should use batteries. KA must effectively resolve the ambiguity and fill in the missing requirement in order to perform the mapping from the requirements specifications in Figure 3 to the functional description in Figure 4.

FUNCTIONAL SPECIFICATION

STIMULUS: SUBSTANCE: force
LOCATION: switch

INPUT: SUBSTANCE: electricity
LOCATION: battery

OUTPUT: SUBSTANCE: light
INTENSITY: 18 lumens
COLOR: blue

FIGURE 4. The output for Text 1—SBF functional specification.

5.2. THE PROCESS

Briefly, KA achieves this mapping by performing the following actions iteratively. First, it reads a text word by word and sentence by sentence, building a syntactic and conceptual interpretation of the text. Structural and lexical ambiguities encountered along the way are resolved by combining information from lexical, syntactic and conceptual knowledge. The results of this language process is a representation of the meaning of the text in the ontology of the domain captured by the SBF language. For the text in Figure 3, for example, the interpretation is a representation of a *tentative* functional specification of the device.

Second, the functional specification is sent to the memory process and the comprehension process. The memory process searches the case memory and retrieves a set of cases which at least partially match the tentative functional specification. These retrieved cases are sent to the comprehension process. The comprehension process uses the differences between the tentative specification of the new device and the specification of the retrieved cases (if any) to provide feedback to the language process. This feedback is in terms of the differences between the two specifications.

Third, this feedback is sent to the language process. The language process combines the feedback with its current tentative interpretation, filling missing details. The parts of the current interpretation that are inconsistent with the feedback are re-examined and other alternatives are considered. It is in this way that the recovery from erroneous decisions in the resolution of ambiguities can be made. The language process communicates the results of its decision-making in the form of a new functional specification.

Once this text interpretation is consistent with the design experience, a complete interpretation is produced and sent to the problem solver. Below, we discuss these steps in further detail.

5.2.1. Producing a tentative interpretation

The language process begins by performing a syntactic parse of a sentence in the input. Parsing resolves any ambiguities in the word's syntactic categories and the sentence's syntactic structure. Once this parse has been completed, the concepts denoted by the content words[†] found in the lexicon are sent to the semantic network. Choosing the syntactic categories of words, the parser, in effect, selects the word meanings that will be considered by conceptual processing. Only those concepts that are consistent with the syntactic categories chosen are sent to the semantic network.

After receiving all of the concepts denoted by the content words in the sentence, the semantic network begins by activating a semantic node for each concept. Since lexically-ambiguous words such as "input" denote multiple concepts, multiple nodes are activated by the appearance of words such as "input" in the sentence.

[†] Words are partitioned into two classes: *Function words* and *Content words*. Function words (e.g. Prepositions, Articles, Conjunctions, etc.) are also called *closed class words* because the addition of new words to the class is rare. Content words (e.g. Nouns, Verbs, etc.) are also called *open class words* because the addition of new words occurs frequently.

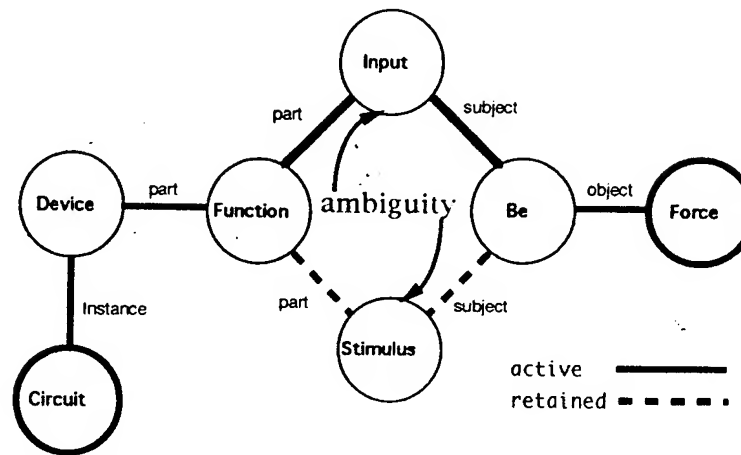


FIGURE 5. Resolving ambiguity in the semantic network.

The semantic network identifies relevant conceptual relations between these active concepts (nodes) by *marker passing*, a standard method of inference used in semantic networks (Charniak, 1981; Hendler, 1986; Norvig, 1989). *Marker passing* identifies complex conceptual relations (paths) in the semantic network that connect active concepts (nodes), producing them as inferences. *Marker passing* is achieved by (1) initializing a marker for each active node, (2) sending copies of the marker to all of the nodes that maintain semantic links with the active node, and (3) continuing semantic link traversal until a maximum path length is reached. Semantic links correspond to primitive conceptual relations between concepts (e.g. part-whole relations, instance relations, property relations) and are the basic elements from which complex conceptual relations are formed. Inferences are produced when "marked" sequences of primitive conceptual relations (paths) that connect active nodes are identified. For example, in the network in Figure 5, an inference is generated for the "marked" sequence (Circuit, instance, Device, part, Function, part, Input, subject, Be, object, Force) which connects the active nodes **Circuit** and **Force**.

After a set of inferences (paths) has been proposed by *marker passing*, the semantic network begins resolving ambiguities in the interpretation. Ambiguities are marked by the words that evoke them. For example, "input" specifies that any interpretation may include either the concept **input** or the concept **stimulus** but cannot include both. In other words, either the node **input** or the node **stimulus** can appear in the paths that make up the final interpretation, but both cannot. These ambiguities are resolved by *consistency-checking*.[†] Consistency-checking is done in accordance with the SBF ontology. The semantic network identifies inferences that are deemed inconsistent by its SBF representation, resolves the inconsistency in favor of the inference that has the most in common with the current interpretation, and places the other on the retained list. Retained inferences can be recalled if the situation warrants it.

[†] See Wilks (1973) for a discussion of how semantic consistency-checking may be used to resolve ambiguity.

To illustrate how this works, consider the network in Figure 5. In this network, there are two inferences that relate the active concepts **Circuit** and **Force**. Each contains a concept denoted by "input". The path (**Circuit**, instance, Device, part, Function, part, Input, subject, Be, object, **Force**) contains the concept **Input**, and the path (**Circuit**, instance, Device, part, Function, part, Stimulus, subject, Be, object, **Force**) contains **Stimulus**. As defined by the SBF ontology and representation, a force cannot be both the input and stimulus to the circuit, so these inferences are deemed inconsistent. The semantic network recognizes this by noting that the paths relate the same two active nodes. This simple method of inconsistency recognition is made possible by the fact that the network is specified in such a way that it conforms to the SBF ontology. The network resolves this ambiguity in favor of the **input** inference because a number of inferences proposed by the network involve the concept of **input** but only a few include **stimulus**. The language process has a set of such heuristics for resolving semantic ambiguities by selecting between paths in the semantic network (Eiselt, 1989). Finally, **input** is kept in the current interpretation, and the inference containing **stimulus** is retained as noted by Figure 5.

Once consistency-checking has been completed and the ambiguities resolved, the inferences articulate a tentative functional specification which is consistent, where consistency is defined by the principles of the SBF representation and ontology.

5.2.2. Model-based retrieval and comprehension

The tentative functional specification produced by the language process is sent to both the memory and comprehension process. The memory process searches the case/model memory for a case that most closely matches the given specification. Given the tentative specification discussed in this example, the memory process finds a case that describes a device producing light with color blue and intensity 8 lumens. This is a partial match of the desired device since the specifications differ only in the intensity of light produced. This case is sent to the comprehension process.

The comprehension process compares the tentative specification and the retrieved specification and notes the difference in light intensity. In an attempt to explain what about the behavior of the device causes this difference in intensity, the comprehension process performs a diagnosis on the retrieved case to determine the factors which contribute to the intensity of the output. In doing this diagnosis, the comprehension process uses the causal model of the device, that is, the behaviors that are included in the SBF description of this device.

During the diagnosis, the comprehension process notes the inconsistency between the input described in the retrieved case, "Electricity", and the input described in the functional specification produced by the language process, "force on a switch". It suggests to the language process that it is more likely that the "input is electricity". It also feeds back the suggested functional requirement that "electricity is provided by a battery".

5.2.3. Recovery from error

The language process accepts feedback from the comprehension process and attempts to incorporate it into its interpretation. For example, in the case of the suggestion that the "input is electricity", it activates the concept **electricity** and the path of conceptual relations that connect **electricity** to **input**. This is illustrated in

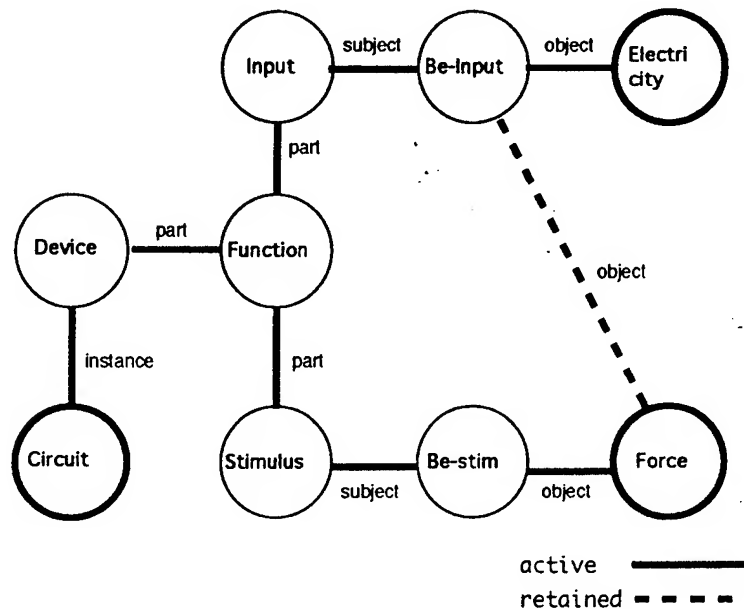


FIGURE 6. Recovering from error in the semantic network.

Figure 6. Once the feedback has been activated, the semantic network checks the consistency of the new interpretation and resolves any conflicting inferences.

Consider how this works given the network in Figure 6. The semantic network identifies an inconsistency between the "input is force" inference denoted by the path that travels from **Circuit** through **Input** to **Force** and the "input is electricity" inference denoted by the path from **Circuit** through **Input** to **Electricity**. Under the SBF representation, devices may have only one input. Both this inconsistency and the inconsistency between the retained inference "stimulus is force" and the "input is force" inference serve to make the "input is force" inference unlikely. To recover from this erroneous inference, the semantic network places the "input is force" inference on the retained list, recalls the "stimulus is force" inference, and keeps the "input is electricity" active, successfully resolving the lexically ambiguous word "input". In a similar way, the inference "electricity is provided by a battery" is also incorporated into the current interpretation.

Finally, the language process sends the new functional specification to the comprehension process, which completes its diagnosis and produces a design that satisfies the new functional description of the device.

6. Investigation 2

In the second investigation, we examined whether KA's design situation and problem-solving methods could help infer the relevance of indirect statements as well as identify relationships between design details underspecified in the natural language surface form. In this investigation, we sought to take advantage of (1) KA's memory of case-specific SBF models, (2) KA's model-based adaptation

The system shall consist of two computer elements interfaced to each other over an XXXX link. Computer A shall send a K byte request packet to Computer B every M seconds. In response to the request packet Computer B shall send a L byte response packet back to Computer A. Packet encoding is N bit ASCII.

FIGURE 7. Text 2—sample requirements specification.

capability, and (3) KA's model-based diagnosis capability in extracting both a functional and a structural specification from a requirements specification. Our results indicate the following benefits:

- Using case-specific SBF models as the starting point for the interpretation of a requirements specification enables the language process to identify the relevance of statements that, on the surface, appear to be irrelevant to the design requirements.
- Model-based adaptation prevents missing "the big picture" by fashioning a functional specification from a disparate set of requirements that do not directly make statements about the function of the device to be designed.
- Using KA's SBF models and diagnosis capability ensures that critical relationships between design details that are left unarticulated in the written requirements are identified and that these relations impact the structural specification extracted from the text.

6.1. THE TASK

In the current investigation, we focused on extracting the critical features from ill-specified texts such as that in Figure 7.[†] Its corresponding SBF description appears in Figure 8.

FUNCTIONAL SPECIFICATION

INPUT: SUBSTANCE: request-message
SIZE: K byte

OUTPUT: SUBSTANCE: response-message
SIZE: L byte

STRUCTURE
COMPONENT: computer A

COMPONENT: computer B

COMPONENT: link
BAUD-RATE: Z

FIGURE 8. The output for Text 2—SBF functional specification.

[†] Specific design details have been masked to protect proprietary information of our sponsor, Northern Telecom, who supplied this example as a test case.

To successfully understand the passage in Figure 7 as a design specification, one must be able to determine the function of the device being described, its inputs, and its outputs. However, none of these characteristics are explicitly described in the text. The text describes the device (referred to as "the system") in terms of its components (e.g. "computer A", "computer B"), their connectivity (i.e. "Computer A shall send... to Computer B") and types of information they transmit (e.g. "request packet"). Nothing is stated about the function of "the system". Its inputs and outputs are not even referred to. To achieve the mapping from the English description in Figure 7 to the functional specification in Figure 8, KA must be able to generate functional requirements from alternate information sources, using the information provided in the text (e.g. K byte request packet, L byte response packet) as constraints on the generation.

Before KA can use this information to constrain generation, however, it must determine its relevance to the function of the device. In the text, it is unclear what a "request packet" and a "response packet" have to do with the function of the device. On the face of it, they appear to have no relevance to the device's inputs and outputs, being simply relegated to its internal workings. An understanding of "requests" and "responses" reveals, however, that these information packets are the inputs and outputs of the device, as indicated in Figure 8. If KA is to make use of such indirect statements about functional requirements, it must use its specific, episodic knowledge about computing networks to infer that a *request* message passed from one computer to another is the input to the system, and the message sent in *response* is the output.

Although the text devotes significant attention to design details such as the frequency of transmission "every M seconds", the size of the request packet "K byte", and the size of the response packet "L byte", it leaves the relationship between these details unspecified. Identifying the relationships between these design details is critical to producing a successful design. Using the frequency of transmission in combination with the size of the information packets, KA can infer the appropriate baud rate for the link between the system's two computers. Without this baud rate specification, KA may select a link that is too slow, producing an unusable design, or it may select a link that is much faster than needed, producing an expensive and possibly unbuildable design. If a competent interpretation of this text and a successful design solution are to be produced, these design details must be combined into a coherent specification of the "XXXX" link's baud rate.

6.2. THE PROCESS

To produce the specification in Figure 8 from the text in Figure 7, KA must generate a functional specification that is constrained by the information provided in the text. To make use of this information, KA must infer the relevance of indirect statements, and combine detailed design requirements into coherent specifications.

Briefly, KA performs the mapping from the text in Figure 7 to the functional specification in Figure 8 in the following manner. First, using its memory of past design cases and case-specific SBF models, KA employs a complete SBF model as a baseline from which the relevance of indirect statements about the function of the device can be inferred. The memory process extracts a relevant model from its

case/model memory using the bits and pieces of a tentative interpretation produced by the language process and sends it to the comprehension process. This model is fed back to the language process. Using this model, as baseline, the language process employs its inference generation capability (i.e. marker passing) to identify the relations between the feedback and the concepts specified by the text. Once the language process has finished its inference generation, it produces a tentative functional specification of the design which is sent to the comprehension process.

Second, KA performs model-based adaptation on the SBF model, generating a new case-specific SBF model that is consistent with the information provided in the tentative functional specification. The comprehension process identifies distinctions between the tentative functional specification and the SBF model. Then, it uses these distinctions to modify the SBF model. During adaptation, the comprehension process modifies only those aspects of the stored model that conflict with the tentative specification. This leaves a significant number of design details unaffected. In effect, design details are transferred from the stored SBF model to the new device model.

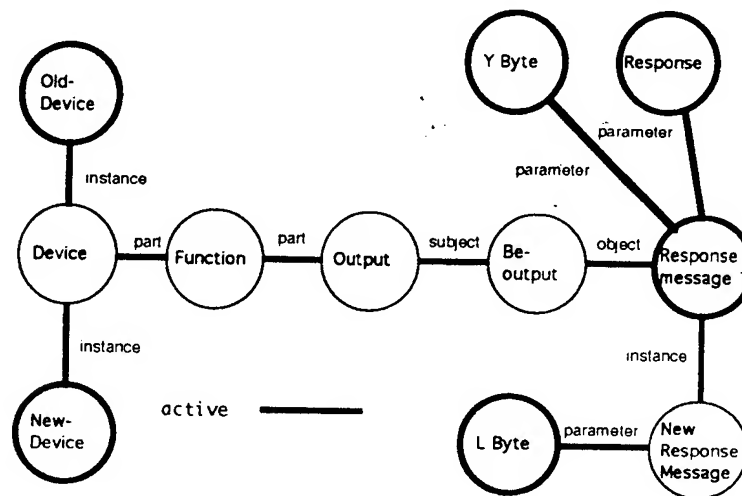
Third, during adaptation, KA identifies those distinctions that require changes to the new device's structure and adapts the tentative design specification accordingly. The comprehension process notes differences between the stored device model and the tentative functional specification, such as a difference in output and input that may require changes in the structure of the new device. To accommodate these changes, it selects generic modification plans that modify the device structure. Model-based diagnosis is performed on the stored device model, and the necessary modifications to the device structure are determined. Using the products of model-based diagnosis and the selected generic modification plans, the comprehension process adapts the tentative design specification such that it includes a structural description that is consistent with the functional specification.

Once this process of adaptation is completed, the new design is sent to the problem-solving process for verification and possibly further design. Then this new case is stored in KA's memory of case-specific models for later reuse. Below, we discuss each of these steps in detail.

6.2.1. Inferring the relevance of indirect statements

The memory process begins by sending a relevant SBF model to the comprehension process which feeds it back to the semantic network in the language process. The semantic network activates the model's corresponding concepts and conceptual relations. For example, in the subsection of the semantic network displayed in Figure 9, the concepts **Old-Device**, **Y Byte**, **Response message** and **Response** and the primitive conceptual relations that relate these concepts (e.g. *parameter*, *instance*, *part*) are activated by feedback from the Comprehension process.

Much like the previous example, the input is then parsed and the content words of each sentence are passed to the semantic network, which initiates marker passing at each word's corresponding concept. Using the feedback as a bridge, the semantic network identifies conceptual relations between the concepts activated by the text and constructs a new set of inferences. The new inferences relate concepts specified in the text to the functional specification of the new device. Finally, a tentative

FIGURE 9. Identifying the relevance of *response* and *L Byte*.

functional specification of the new device is produced from these new inferences and sent to the comprehension process.

To see how the feedback acts as a bridge between the concepts activated by the text, consider the subsection of the semantic network displayed in Figure 9. In this semantic network, the concept **L Byte** is activated by the appearance of "L Byte" in the input text in Figure 7, **Response** is activated by the appearance of "response", and **New-Device** is activated by the appearance of "system". Using only these active concepts, the semantic network would be *unable* to identify critical conceptual relations such as those between **L byte** and the **Output** of the **New Device** because the active concepts **L byte** and **New Device** are only distantly related to each other. Basing its decision on the length of the path between the two concepts, the semantic network would deem it unlikely that the text intends to relate these concepts without further evidence.

However, when the semantic network begins with feedback-activated concepts such as **Response message**, conceptual relations such as those between **L byte** and **New Device**, as well as those between **Response** and **New Device**, can be identified. The concepts activated by the text are more closely related to the concepts activated by feedback than they are to each other, so the semantic network can identify conceptual relations between the text-activated concepts and feedback-activated concepts (as indicated by the active paths in Figure 9). This produces inferences that serve to relate the text-activated concepts, inferences that identify the relevance of concepts such as **Response** and **L byte** to the function of the **New device**.

6.2.2. Generating a new functional specification

Given a tentative and underspecified functional specification produced by the language process, the comprehension process compares the SBF model and this underspecified functional specification to determine the distinctions between the two device descriptions. It notes distinctions that are extremely significant, such as the

distinction between the sizes of the response package (*L bytes* vs. *J bytes*), and those that are less significant, such as the difference in the names of the components (*A* vs. *C*).

Once all of these distinctions have been collected, the comprehension process begins adapting the stored SBF model. It modifies the component names such that they are consistent with the new functional specification, changes the sizes of the response package and request package, etc. In doing so, it transfers a large amount of the SBF model of the known device to the SBF model of the new device. For example, it transfers the types of the components in the old device to components of the new device. The result is that all of the design details are filled in, and a significant number of assumptions are made. The comprehension process assumes that the new device has the same behavioral descriptions as the stored device and the same structural description.

6.2.3. *Identifying relationships between design details*

During the adaptation of the stored SBF model, the assumption that the structural specifications of the new and stored designs are equivalent is examined. The comprehension process considers each of the differences it has identified between the new specification and the stored specification, looking for those differences that may require modifications to the device structure. Differences that are particularly relevant are differences in device inputs and outputs. For example, in this example, the distinction between the size of the new design's output and the stored design's output (i.e. *L bytes* vs. *J bytes*) imposes new constraints on the structure of the new design. The comprehension process collects these differences and orders them with respect to their priority.

Examining them in order of their priority, the comprehension process retrieves generic modification plans that rectify the differences between the new design specification and the stored design specification by adapting the stored SBF model. Generic modification plans are selected by the type of differences they reduce. In achieving their ends, generic modification plans manipulate, delete and augment device structure. They include component replacement, substance substitution, parametric modification, component deletion and component insertion and cascading.

After the comprehension process has received the generic modification plans from the memory process, it begins to diagnose the new model's failure, in this particular example, its failure to produce the desired output. It is during diagnosis that the comprehension process recognizes the relationship between the design details *every M seconds* and *L byte response packet*. The comprehension process correctly assumes that since the new design specification identifies an output which differs from the stored specification, the new design's current structural specification will fail to produce its specified output. The design produces the output of the stored design because the comprehension process has assumed that they have equivalent structural specifications. While investigating the causes of the new design's failure, the comprehension process identifies the relationship between the frequency of transmission (i.e. *every M seconds*), the size of the response packet (i.e. *L bytes*) and the baud rate of the link component. Using the qualitative relations specified in the stored SBF model, it notes that the baud rate of the link component limits the

amount of information that can be transferred at a particular frequency. It concludes that the baud rate of the current link component is too low and that increasing the baud rate of this component would provide for the size of the response packet in the new design and the desired frequency of transmission.

Given the diagnosis and the generic modification plans, the comprehension process "repairs" the structural specification of the new design, using component replacement. It replaces the link component in the stored SBF model with a link component that has a higher baud rate. This modification appears in the specification of structure in Figure 8.

7. Investigation 3

Natural language texts are used to achieve a variety of communication goals at different stages in the design process. For example, at later stages in the design, customers use English texts to communicate feedback to designers. Understanding these texts is essential for redesigning a product to meet customer needs. In keeping with our goal of "situating" natural language within design, our third and most recent investigation examined whether KA could interpret design feedback as well as design requirements.

We chose a problem in the design of a reaction wheel assembly for the Hubble space telescope and examined KA's ability to understand user feedback. We demonstrated that KA was in fact able to understand such user feedback and use the information it could extract from the feedback to redesign the reaction wheel assembly. This cost effective redesign was made possible by:

1. KA's repertoire of plans for incremental redesign to correct the malfunction; and
2. KA's ability to precisely identify the part of the design that is to blame for the malfunction.

This investigation demonstrated that KA's theory of situated natural language understanding that was effective in understanding initial design requirements also serves to accomplish interaction with the customer and iterative redesign to meet customer requirements. This successful demonstration shows the extensibility of KA's theory of situated understanding.

7.1. THE TASK

In this investigation, we focused on interpreting and acting upon user feedback. The task is to redesign a malfunctioning component given feedback written in English text. This overall task decomposes into language interpretation and redesign. Given a description of an undesired output of the device and a model of the device's structure, function and behavior, the redesign task is to modify the structure of the device so that it does not produce the undesired output. So as to successfully interpret a passage as a redesign problem, the system must be able to identify the

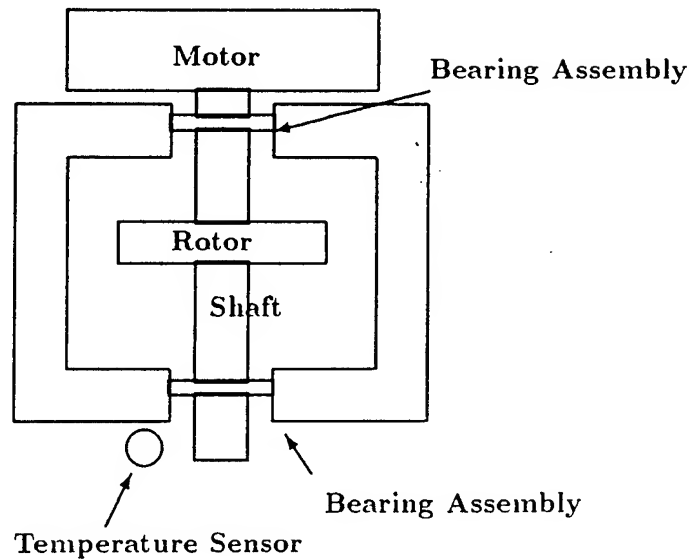


FIGURE 10. The reaction wheel assembly.

relevant device and produce a description of the undesired output. A description of the undesired output includes identifying the device component that produces this output and a description of the output in terms of the relevant substance properties.

Let us consider the specific problem we examined in this third investigation: redesigning the Reaction Wheel Assembly (RWA) aboard the Hubble Space Telescope (Keller, Manago, Nayak & Rua, 1988). The Hubble Space Telescope contains four RWAs; a small portion of one is shown in Figure 10. The desired function of the RWA is to make the telescope point at a chosen area of the universe. The structure of the RWA consists of a rapidly spinning rotor mounted on a shaft. The functioning of the RWA is based on the law of conservation of angular momentum. The direction of the telescope is changed via a signal from Earth sent to the motor which changes the amount of power supplied to the motor. This causes a change in the motor's angular momentum which in turn affects the angular momentum and angular velocity of the shaft. Due to the conservation of angular momentum, the angular momentum of the telescope as a whole changes in the opposite direction. When the telescope nears its desired orientation, a change in the angular momentum of the telescope in the opposite direction reduces the telescope's angular velocity to zero. The vector sums of the angular momentum imparted by the four RWAs enable a rotation of the telescope about any axis.

A common problem in the operation of devices like the RWA arises due to friction in the bearing assemblies. The load on the bearings due to the rapid spin of the rotor causes deformation of the bearing balls which results in increased frictional forces in the bearing assembly. This causes generation of heat in the bearing assembly. The increased temperature in the bearing assembly is an extremely undesirable behavior.

In our third investigation, we examined this redesign problem, taking the text in

The ball bearings in the RWA generate an excessive amount of heat.

FIGURE 11. Text 3—sample requirements specification.

Figure 11 as the typical type of feedback that a user would provide. To produce the appropriate redesign specification given the sentence in Figure 11, KA must identify that:

1. The relevant device is the RWA.
2. The components producing the undesired output are the ball bearings.
3. The undesired output is heat, whose magnitude is too high.

7.2. THE PROCESS

KA performs the mapping from the user feedback in Figure 11 to the specification of the undesired behavior. It reads the sentence word by word and builds a syntactic and conceptual interpretation of the sentence. The result of this language process is a representation of the meaning of the text in terms of the SBF language.

Next, diagnosis determines the component parameter that is responsible for the undesired behavior and the parameter modification desired. In the RWA example, the diagnosis accepts as input a specification of excessive heat at the bearing assembly and returns as output the component parameter, the size of the ball bearings, and the parameter modification desired, an increase in the size of the ball bearings.

Third, the repair task replaces the component with a component that has the desired parameter value. In the RWA example, this task replaces the old ball bearings, thereby redesigning the RWA to eliminate the problem of excessive heat in the bearings, which was noted by the user in the feedback.

8. Discussion

The current implementation of KA is able to extract both functional and structural specifications from design requirements or from user feedback written in English. To do so effectively, it overcomes many of the difficult problems that face any natural language understanding system. It can resolve semantic and syntactic ambiguities, correctly infer unarticulated statements, identify the relevance of indirect statements, determine the unspecified topic/theme of a passage from its constituent statements, and combine disparate statements into coherent interpretations.

We have tested the KA system with examples of design specifications and user feedback in several different domains, including electrical circuits, computer networks and mechanical dynamic systems. Many of the texts used in these tests, such as the one in Investigation 2, came from real-life examples from the industry. We have investigated similar domains in our ongoing work on design (Goel, 1991a, b; Bhatta & Goel, 1992, 1993), where we have demonstrated that our design

system can acquire knowledge about new devices in the form of cases and models as a result of design problem solving. Since KA has been shown to provide solutions to problems (such as ambiguity) in understanding design specifications of new (yet related) devices by using such previously acquired design knowledge, its methods for language understanding in the design situation show a strong promise of bootstrapping and scalability.

Apart from providing a new approach to attacking the natural language understanding problem, the task that KA addresses is also a problem of high practical relevance. There is a great need to build automated systems that can take design specifications for both physical hardware and software "devices" and represent them in formal ontologies that are comprehensible to the designers no matter whether they are human design teams or automated design systems or computerized design aids. KA is a demonstration that it is possible to build such systems. In addition, KA's ability to infer unarticulated requirements from its cases and models makes it a useful model of making sure that the "common" design knowledge that is supposed to be shared between customers and designers in each domain is included in the design process, thereby reducing customer dissatisfaction and the need for expensive redesign at later stages.

8.1. A CRITIQUE OF KA

However, our work also raises certain issues, the addressing of which is part of ongoing work in the KA project. These issues are representative of the open problems that exist in the field of natural language understanding and, more generally, artificial intelligence. They involve limitations in both the input that KA can accept and the outputs that it can produce. The system's representations also have some limitations. In addition, restrictions on interaction limit the extent to which system components can avail themselves of the system's resources. In the hope of clearly elucidating the capabilities of our work, as well as its limitations, in this section we discuss each of these problems and their impact. We close with a short summary of recent work within our research group that has sought to remedy these problems and outline our research plans for addressing them in the future.

8.1.1. *Input*

While KA is able to master many of natural language's lexical and structural ambiguities and ascertain aspects of meaning that are left unarticulated in texts, certain classes of natural language discourse remain beyond its reach. KA's method of resolving ambiguity relies on a text that is internally consistent. Single interpretations are produced when their alternatives are found to be inconsistent with a combination of linguistic evidence and the evidence provided by KA's memory of case-specific models. It is certainly possible, however, for the text to be internally inconsistent. Design specifications may describe a conflicting set of requirements that prevent a consistent interpretation. In this case, KA should communicate these inconsistencies to the user, and possibly pursue an interpretation that reflects a consistent subset of the text's content. Such a capability is beyond the current scope of our work.

8.1.2. *Output*

Although KA is able to produce both functional and structural design specifications given design specifications written in natural language, there are some types of design information that it is unable to deal with. Examples include design requirements, which are typically part of design specifications that customers provide.

8.1.3. *Representation*

The semantic network representation used in the language process gives it the ability to precisely articulate the SBF representation and ontology in such a way that inconsistencies in the interpretations can be easily recognized. However, this precision has its associated cost. The representation affords little generativity. All of the concepts and the relations that hold between them must be explicitly articulated. We would like KA to be able to extract automatically some of this knowledge in the semantic network from the cases and models it acquires through its design problem-solving experiences.

8.1.4. *Interaction*

The KA architecture ensures that productive interaction occurs between the system components. The memory process lends its ability to select the appropriate case-specific device models to the language, comprehension and problem-solving processes; the language process provides cues that assist the comprehension process; both the comprehension and problem-solving processes share many of the same methods. However, the architecture does still maintain a few well-demarcated boundaries that decrease the effectiveness of the system. For example, communication between syntactic parsing and semantic inference is overly restricted. We would like semantic inferences to have a greater effect on syntactic decision-making, and to increase the influence that syntactic decision making has on semantic inferences.

8.2. RECENT WORK

In recent work in natural language processing, we have sought to rectify some of the problems above. Members of our research group have developed a model of sentence understanding that uses semantic inferences to avoid and recover from errors in syntactic parsing (Eiselt, Mahesh & Holbrook, 1993; Mahesh, 1994; Mahesh & Eiselt, 1994). Others have developed a model of sentence understanding that uses syntactic evidence to infer semantic interpretations (Peterson & Billman, 1994).

9. Related work

Our work is related to five different bodies of research in natural language understanding: situated natural understanding, integrated representations for natural language understanding and problem solving, Conceptual Information Processing, the understanding of natural language descriptions of physical devices, and the

modularity of "Mind". Below, we describe the relationships between these bodies of work and our own.

9.1. SITUATED NATURAL LANGUAGE UNDERSTANDING

Our research was inspired in part by Winograd's SHRDLU system (Winograd, 1973), which was one of the first successful systems to situate language understanding in problem solving. SHRDLU formed plans for actions in a simulated blocks world based on its interpretation of external commands expressed in English. It explored certain interactions between language understanding and planning, and demonstrated the methodological usefulness of exploiting the constraints imposed by planning on language understanding, and vice versa. Of course, SHRDLU also suffered from a number of well-known problems. For example, it assumed a closed world, it represented knowledge procedurally, it lacked the capability of abstract reasoning, and it also lacked sufficient control over processing.

Since the construction of the SHRDLU system in the late 1960s, research in Artificial Intelligence has led to a large collection of new results in the areas of representation of knowledge and control of reasoning. For example, languages for descriptively and explicitly representing models of a physical situation, and methods for revising stored models to meet the specifications of new situations, are of relatively recent origin. We believe that the necessary technologies are now ripe for once again investigating situated language processing in the context of problem solving. Specifically, our research seeks to explore and exploit the use of the design situation for natural language understanding.

9.2. INTEGRATED REPRESENTATIONS FOR NATURAL LANGUAGE UNDERSTANDING AND PROBLEM SOLVING

Several attempts have been made to integrate natural language and problem solving using a common representation for both language comprehension and reasoning (Rieger, 1976; Simon & Hayes, 1979; Charniak, 1981; Wilensky, 1983; Beck & Fishwick, 1989). Our work continues in this direction by applying functional models and reasoning to the understanding process. The same ontology is used for understanding natural language and reasoning about devices. It is noteworthy, however, that in the past, common representations for language understanding and problem solving have generally implied a unified process for the two tasks. That is, the same process over the same representations is used for both language-understanding and problem-solving tasks. Our work differs in that, while we use a common ontology, the processes for language understanding and problem solving are not identical nor even equivalent. While the language-understanding and problem-solving tasks in KA support each other and share some subtasks and subprocesses (e.g. memory retrieval), they are distinct in the subtasks they set up and the processes they use.

The goals of Grishman's PROTEUS system (Ksiezzyk & Grishman, 1989), which comprehends failure reports, are not unlike our own goals. PROTEUS, however, did not implement diagnosis and repair. More importantly, language understanding in PROTEUS is driven merely by the templates they wish to fill. We are developing

a more general theory of language and applying it to extract the information we need for the design process.

9.3. CONCEPTUAL INFORMATION PROCESSING

Many language understanding theories have used high-level knowledge structures to guide the understanding process. Schank and Abelson (1977), for example, described the use of stored scripts. The script theory represented knowledge about stories as well as story interpretations in terms of temporally-ordered sets of events. It was employed in a story understanding system called SAM (for Script-Applier Mechanism; Cullingford, 1978). SAM identified the temporal relation between two events by assuming that the linear sequence of sentences in a story corresponded to the temporal ordering of events. SAM's ability to apply scripts and produce interpretations depended critically upon this seemingly simple assumption. In KA, we make no such assumption about temporal correspondences between the discourse and the knowledge structures.

In her work, Lehnert proposed an object representations called "Object Primitives" which assist in making inferences about objects described in natural language texts (Lehnert, 1978). Although there is merit in this object-centered representation, in our work we have found causal relations between substances and components as well as the casual behaviors of devices to be much more effective aids in resolving problems in natural language understanding.

Other work in conceptual information processing has proposed theories of language understanding with an even stronger reliance on specific knowledge structures (Wilensky, 1978; Lebowitz, 1980; Dejong, 1983; Ram, 1989). While these systems have demonstrated deep understanding abilities in small domains, they have not shown how each of the many types of knowledge structures they need for language understanding (Lehnert, 1978; Martin, 1990) can be acquired without being hand-coded. As a result, this class of systems for language understanding shows little promise for bootstrapping or scalability.

9.4. UNDERSTANDING NATURAL LANGUAGE DESCRIPTIONS OF PHYSICAL SYSTEMS

While we believe that our approach to language understanding in the design situation is quite novel, it is not the first time that researchers have tied text understanding to models of physical systems. Lebowitz's RESEARCHER (Lebowitz, 1983), for example, read natural language texts in the form of patent abstracts, specifically disk drive patents, and updated its long-term memory with generalizations made from these texts. What RESEARCHER stored in its memory was a generalized representation of a disk drive, consisting of a *topographic model* of the disk drive which specified its components and the topographic relationships among them. RESEARCHER's knowledge representation scheme was oriented toward objects and their topographic relationships, which was a departure from most natural language understanding systems of that time, which had typically focused on actors, events and causal relationships. RESEARCHER then used this knowledge to aid in the top-down understanding of additional patent texts. However, RESEARCHER's emphasis on components and topographic relationships left it unable to build causal models of the mechanisms described. In other words,

RESEARCHER effectively knew how a disk drive was constructed, but it did not know how it worked.

Dyer, Hodges and Flowers (1987) and Hodges (1993) describe EDCA, a conceptual analyser which serves as a natural language front-end for EDISON, a naive design problem solver. EDCA uses knowledge of the function of physical devices to produce an episodic description of a device's behavior as described by an input text. This episodic description can then be used to generate a new device model to be integrated into long-term memory. The result is a much more comprehensive understanding of the device's functionality than was possible with RESEARCHER, but EDCA's analysis of the device description is not fully integrated with the process for generating new device models and incorporating them into memory. EDCA, in other words, is but a front-end to EDISON.

As Selfridge (1989) notes, separating the process of analysing the input from generating and incorporating the new model is misguided—the process of understanding a device description *is* the process of building and incorporating a causal model of that device. This is the approach that we have followed in our work, and this approach led us to the KA system which, we believe, corrects the shortcomings of both RESEARCHER and EDCA.

9.5. MODULARITY OF MIND

As well as providing a viable model for solving problems in natural language understanding, our work also addresses a contentious issue in cognitive science, viz. the modularity of "mind". Although it seems clear that language understanding requires cognitive abilities far beyond those that pure linguistic knowledge permits, it is unclear in what manner, if any, linguistic and non-linguistic processes interact. Advocates for the modularity of "mind" have argued for a very limited form of interaction (Fodor, 1983; Jackendoff, 1987). Others have contended that the interaction is so open-ended as to make any boundaries between linguistic processing and the other cognitive processes insignificant (Marslen-Wilson & Tyler, 1989). We propose a modular processing architecture that contains separate language-understanding and problem-solving components. These components interact in at least two significant ways. They share common knowledge, and they communicate the results of their reasoning to each other.

What lessons regarding the modularity of "mind", even tentative ones, can be drawn from our work on KA? KA certainly is modular, but the nature of the modularity depends on the level at which it is analysed. Modularity in KA can be viewed at the levels of task, process and knowledge, and representation. At the task level, "language processing" and "problem solving" are distinct modules, characterized by the types of information they take as input and give as output. At the next level, some of the processes are task-specific but others are shared. Language processing and problem solving, for example, are both informed by the same memory processes which retrieve episodic and conceptual information. Similarly, some of the knowledge is task-specific and some of it is shared. Only the language processes use lexical and syntactic knowledge, and only the problem-solving processes use knowledge of the repair plans used for redesigning devices. On the other hand, both the language and problem-solving processes employ functional

and causal knowledge of devices. Finally, at the level of representation, the language and problem-solving processes share the same SBF ontology for representing conceptual knowledge. Thus, from the viewpoint of KA, the issue of modularity is much more complex than either the orthodox "modularists", such as Fodor and Jackendoff, or the "non-modularists", such as Marlen-Wilson and Tyler, suggest.

10. Conclusions

At one level of abstraction, our work on KA leads to the following conclusions regarding the use of the design situation for natural language understanding:

- Ontologies and knowledge structures available in the design situation (and in intelligent design systems) can be used to resolve ambiguities in natural language inputs to the design system.
- Unarticulated design requirements can be inferred from past design problems described in case-specific SBF models.
- The relevance of indirect statements to design requirements can be inferred by using case-specific SBF models as the starting point for the interpretation of a requirements specification.
- A coherent functional specification can be produced from a disparate set of written structural requirements by applying model-based adaptation to case-specific SBF models.
- The above solutions can all be implemented in an integrated yet modular architecture in which different "modules" interact with each other by communicating their results and decisions with each other. Natural language understanding requires such an iterative cooperation between language-specific and non-linguistic reasoning processes.
- All of the knowledge structures needed for the above solutions (apart from the basic ontology of devices and the lexical entries and meanings of new words) can be acquired from previous design problem solving experiences of the system.

Our work on KA provides solutions to the classical problems of language understanding. Building useful systems with an ability to understand "real" natural language input has been an elusive goal for many years now. Well-known problems, such as ambiguity, indirectness and incompleteness of natural language inputs, have thwarted efforts to build natural language front-ends to intelligent systems. KA solves these problems by exploiting the knowledge and problem-solving processes in the situation of designing simple physical devices. In addition, since the types of knowledge structures used by KA can in fact be acquired by the results of the very problem-solving processes that the system performs, we have shown that KA can be bootstrapped to understand design specifications and user feedback about new devices using the knowledge structures it acquired for similar devices it designed previously. We strongly believe that such an approach to situated natural language understanding, where a problem-solving situation, such as device design, provides the knowledge that is needed for language understanding, is the right way to pursue research in building scalable intelligent systems with useful abilities to interact in a natural language.

This work has been supported by the National Science Foundation (research grant IRI-92-10925), the Office of Naval Research (research contract N00014-92-J-1234), and Northern Telecom (research gift). The authors would like to thank their colleagues Sambasiva Bhatta, Andres Gomez de Silva Garza, Kurt Eiselt, Jeff Pittges and Eleni Stroulia for their efforts and their suggestions and criticisms on the KA project.

References

- BECK, H. & FISHWICK, P. (1989). Incorporating natural language descriptions into modeling and simulation. *Simulation*, 52(3), 102-109.
- BHATTA, S. & GOEL, A. (1992). Use of mental models for constraining index learning in experience-based design. *Proceedings of the AAAI Workshop on Constraining Learning with Prior Knowledge*, pp. 1-10, San Jose, CA, July.
- BHATTA, S. & GOEL, A. (1993). Model-based learning of structural indices to design cases. *Proceedings of the IJCAI workshop on "Reuse of Designs: An Interdisciplinary Cognitive Approach"*, pp. A1-A13, Chambery, France, August.
- BYLANDER, T. (1991). A theory of consolidation for reasoning about devices. *International Journal of Man-Machine Studies*, 35(4), 467-489.
- BYLANDER, T. & CHANDRASEKARAN, B. (1985). Understanding behavior using consolidation. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 450-454.
- CHANDRASEKARAN, B., GOEL, A. & IWASAKI, Y. (1993). Functional representation as design rationale. *IEEE Computer*, January, 48-56.
- CHARNIAK, E. (1981). A common representation for problem-solving and language-comprehension information. *Artificial Intelligence*, 16, 225-255.
- CHARNIAK, E. & McDERMOTT, D. (1985). *Introduction to Artificial Intelligence*. Reading, MA: Addison-Wesley.
- CHOMSKY, N. (1957). *Syntactic Structures*. The Hague: Mouton.
- CULLINGFORD, R. (1978). *Script application: computer understanding of newspaper stories*. Ph.D. Thesis, Yale University, Department of Computer Science, USA.
- DEJONG, G. (1983). *Skimming stories in real time: an experiment in integrated understanding*. Ph.D. Thesis, Yale University, Department of Computer Science, USA.
- DYER, M., HODGES, J. & FLOWERS, M. (1987). *Computer comprehension of mechanical device descriptions*. Technical Report UCLA-AI-87-7, University of California, Los Angeles, Artificial Intelligence Laboratory.
- EISELT, K. (1987). Recovering from erroneous inferences. *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 540-544.
- EISELT, K. (1989). *Inference processing and error recovery in sentence understanding*. Ph.D. Thesis, University of California, Irvine, Department of Information and Computer Science, USA.
- EISELT, K. P., MAHESH, K. & HOLBROOK, J. K. (1993). Having your cake and eating it too: autonomy and interaction in a model of sentence processing. *Proceedings of the Eleventh National Conference on Artificial Intelligence, AAAI-93*, pp. 380-385, Washington, DC, July.
- FODOR, J. (1983). *Modularity of Mind*. Cambridge, MA: MIT Press.
- GOEL, A. (1989). *Integration of case-based reasoning and model-based reasoning for adaptive design problem solving*. Ph.D. Thesis, Ohio State University, Department of Computer Science, USA.
- GOEL, A. (1991a). A model-based approach to case adaptation. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pp. 143-148, Chicago, IL, August.
- GOEL, A. (1991b). Model revision: a theory of incremental model learning. *Proceedings of the Eight International Conference on Machine Learning*, pp. 605-609, Chicago, IL, June.
- GOEL, A. (1992). Representation of design functions in experience-based design. In D. BROWN, M. WALDRON & H. YOSHIKAWA, Eds. *Intelligent Computer Aided Design*, pp. 283-308. Amsterdam: North-Holland.

- GOEL, A. & CHANDRASEKARAN, B. (1989). Functional representation of designs and redesign problem solving. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 20-25, Detroit, MI, August.
- GOEL, A. & CHANDRASEKARAN, B. (1992). Case-based design: a task analysis. In C. TONG & D. SRIRAM, Eds. *Artificial Intelligence Approaches to Engineering Design, Volume II: Innovative Design* pp. 165-184. San Diego, CA: Academic Press.
- GRISHMAN, R. (1986). *Computational Linguistics*. New York: Cambridge University Press.
- HENDLER, J. (1986). *Integrating market-passing and problem-solving: a spreading activation approach to improved choice in planning*. Technical Report CS-86-01, Brown University, Department of Computer Science.
- HENDRIX, G., SACERDOTI, E., SAGALOWICZ, D. & SLOCUM, J. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 8(3), 105-147.
- HODGES, J. (1993). *Naive mechanics: a computational model for representing and reasoning about simple mechanical devices*. Technical Report UCLA-AI-93-01, University of California, Los Angeles, Department of Computer Science.
- JACKENDOFF, R. (1987). *Consciousness and the Computational Mind*. Cambridge, MA: MIT Press.
- KELLER, R., MANAGO, C., NAYAK, P. & RUA, M. (1988). Large multi-use knowledge bases. (Internal Memo) Knowledge Systems Laboratory, Stanford University.
- KINTSCH, W. (1974). *The Representation of Meaning in Memory*. Hillsdale, NJ: Lawrence Erlbaum.
- KSIEZYK, T. & GRISHMAN, R. (1989). Equipment simulation for language understanding. *International Journal of Expert Systems*, 2(1), 33-78.
- LEBOWITZ, M. (1980). *Generalization and memory in an integrated understanding system*. Ph.D. Thesis, Yale University, Department of Computer Science, USA.
- LEBOWITZ, M. (1983). RESEARCHER: an overview. *Proceedings of the Second National Conference on Artificial Intelligence*, pp. 232-235.
- LEHNERT, W. (1978). *The Process of Question Answering*. Hillsdale, NJ: Lawrence Erlbaum.
- LEHNERT, W., DYER, M., JOHNSON, P., YANG, J. & HARLEY, S. (1983). BORIS—an experiment in in-depth understanding of narratives. *Artificial Intelligence*, 20(1), 15-62.
- MAHESH, K. (1994). Reaping the benefits of interactive syntax and semantics. *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pp. 310-312, Las Cruces, NM, June.
- MAHESH, K. & EISELT, K. (1994). Uniform representations for syntax-semantics arbitration. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA, August.
- MARSLEN-WILSON, W. & TYLER, L. (1989). Against modularity. In J. GARFIELD, Ed. *Modularity in Knowledge Representation and Natural Language Understanding*. Cambridge, MA: MIT Press.
- NORVIG, P. (1989). Marker passing as a weak method for text inferencing. *Cognitive Science*, 13, 569-620.
- PETERSON, J. & BILLMAN, D. (1994). Correspondences between syntactic form and meaning: from anarchy to hierarchy. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA, August.
- PETERSON, J., MAHESH, K., GOEL, A. & EISELT, K. (1994). KA: situating natural language understanding in design problem solving. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA, August.
- RAM, A. (1989). *Question-driven understanding: an integrated theory of story understanding, memory, and learning*. Ph.D. Thesis, Yale University, Department of Computer Science.
- RICH, E. & KNIGHT, K. (1991). *Artificial Intelligence*. New York: McGraw-Hill.
- RIEGER, C. (1976). On the organization of knowledge for problem solving and language comprehension. *Artificial Intelligence*, 7, 89-127.
- SCHANK, R. & ABELSON, R. (1977). *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Lawrence Erlbaum.

- SELFIDGE, M. (1989). Toward a natural language-based causal model acquisition system. In W. HORN, Ed. *Causal AI Models: Steps Toward Applications*, pp. 107-128. Hemisphere Publishing Corporation.
- SEMBUGAMOORTHY, V. & CHANDRASEKARAN, B. (1986). Representation of devices and compilation of diagnostic problem-solving systems. In J. KOLODNER & C. RIESBECK, Eds. *Experience, Memory and Problem-solving*, pp. 47-73. Hillsdale, NJ: Lawrence Erlbaum.
- SIMON, H. & HAYES, J. (1979). *The Understanding Process: Problem Isomorphs*, volume 1. New Haven, CN: Yale University Press.
- STROULIA, E. & GOEL, A. (1992). Generic teleological mechanisms and their use in case adaptation. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 319-324, Bloomington, IN, August.
- STROULIA, E., SHANKAR, M., GOEL, A. & PENBERTHY, L. (1992). A model-based approach to blame-assignment in design. *Proceedings of the Second International Conference on AI in Design*, pp. 519-537, Pittsburgh, PA, June.
- WILENSKY, R. (1978). *Understanding goal-based stories*. Ph.D. Thesis, Yale University. Department of Computer Science, USA.
- WILENSKY, R. (1983). *Planning and Understanding. A Computational Approach to Human Reasoning*. Reading, MA: Addison-Wesley.
- WILKS, Y. (1973). An artificial intelligence approach to machine translation. In R. C. SCHANK & K. M. COLBY, Eds. *Computer Models of Thought and Language*, pp. 114-151. San Francisco, CA: W. H. Freeman.
- WINOGRAD, T. (1973). A procedural model of language understanding. In R. C. SCHANK & K. M. COLBY, Eds. *Computer Models of Thought and Language*, pp. 152-186. San Francisco, CA: W. H. Freeman.
- WINSTON, P. (1992). *Artificial Intelligence*. Reading, MA: Addison-Wesley.

Paper accepted for publication by the Editor, Professor B. R. Gaines.

Computational Support for Collaborative Learning through Generative Problem Solving

N. Hari Narayanan, Cindy E. Hmelo, Valery Petrushin, Wendy C. Newstetter, Mark Guzdial and Janet L. Kolodner

EduTech Institute, College of Computing, Georgia Institute of Technology

Abstract

In this paper we present a vision of computer-supported collaborative learning through solving generative problems - problems that promote open-ended inquiry and have multiple solutions. This vision stems from a novel and evolving approach to collaborative learning that we are developing at the EduTech Institute. This approach is based on the following premises: that learning is facilitated by generative problem solving, collaborative work and use of multiple cases; that learning and skill acquisition need to be, and can be, scaffolded through software; and that a computer environment which integrates a shared and structured electronic workspace with a full variety of functionalities can effectively support all of the above. We describe this approach and the architecture of the corresponding computer environment. This environment is designed to serve three critical functions: provide a shared workspace for students, facilitate inter- and intra-group collaborative work, and make available the tools and resources that students need for problem solving and learning. The software components of the environment that have already been implemented are described. In the final section we frame ongoing and planned research and development efforts in terms of the characteristics desired of such an environment and ways of assessing its impact.

Keywords — case-based methods of instruction, educational groupware, instructional strategies and approaches.

1. Introduction

Too often, classroom instruction provides students with many bits of knowledge that they are never able to assemble and apply in productive ways, particularly outside the classroom walls. One reason for this is the focus of traditional schooling on learning isolated facts in compartmentalized disciplines. Not surprisingly, this

knowledge often cannot be transferred to real-world problems. Theories of constructivism and situated cognition suggest that for learning to be useful the learner needs to be actively involved in constructing new knowledge within meaningful contexts, not merely absorbing it. Furthermore, learning is enhanced by group-oriented collaborative work, reflection and articulation. These are therefore the central premises of a multidisciplinary approach to structuring learning within the context of case-based instruction that we are developing at the EduTech Institute. This approach is called Multiple Case-Based Approach to Generative Environments for Learning (McBAGEL).

Three factors distinguish this approach: (1) The use of generative problems to promote learning. Generative problems are those that motivate open-ended inquiry, whose solutions require synthesis, which have multiple solutions, and which, therefore, promote the generation, evaluation and combination of ideas in the course of problem solving. The type of generative problems that we use are design problems. (2) The use of multiple cases provided by computer-based case libraries as knowledge sources to aid problem solving. (3) The emphasis on software-scaffolded and group-oriented collaborative work in and out of the classroom.

We are designing a computer-based learning environment that we expect students to use as a workspace for conducting work as part of this approach. The architecture of this environment and its components is the main topic of this paper. However, since the environment's role is to support collaborative learning in the context of generative problem solving, a discussion of the approach and the educational philosophy behind it precedes the description of the computer environment. Then, software components of the environment that have already been designed, implemented and used in classrooms of Georgia Tech are described. In the final section we frame ongoing and planned research and development efforts in terms

of the characteristics desired of such an environment and ways of assessing its impact.

2. Educational Framework

Our approach is based on a synthesis of ideas on learning and problem solving from the fields of education, cognitive psychology and artificial intelligence. This approach is based on the following five central tenets.

(1) Learning is enhanced by problem solving.

Learning is more effective when it occurs through activities associated with solving generative problems (e.g., identifying and formulating the problem, generating alternatives, evaluating, decision making, reflecting, and articulating) rather than through transmission models of instruction. Design, by its very nature, is a generative activity. Therefore, *design-oriented problems* are particularly effective for technical domains like engineering and architecture and may well provide effective anchors for math and science learning.

(2) Collaborative work is central to learning.

Students are expected to solve problems and do assignments in groups. Group-oriented work, in and out of the classroom, is important both in facilitating learning and in preparing students for today's multidisciplinary team-oriented workplaces. As students work in collaborative groups, they are forced to articulate and reflect upon their thinking, leading to an appreciation of the importance of distributed cognition [14] as well as enhancing learning and subsequent transfer [3]. Collaborative work allows students to successfully tackle problems more complex than what any one group member could do alone.

(3) Access to multiple cases will facilitate flexible learning. Providing students with access to multiple cases that contain information-rich and contextualized descriptions of specific situations set within the broader context of a course can significantly impact learning and transfer. The availability and use of multiple cases during problem solving facilitates learning new knowledge, and supports the adaptation and transfer of previous solutions to the current problem [11]. It is expected that by revisiting design skills through numerous cases, flexible transfer of these skills will be supported [20]. Intelligent computer-based case libraries can provide students with not only such access but also means of flexibly navigating among cases and parts of cases.

(4) Learning and the acquisition of problem-solving skills need to be scaffolded. The experiences implementing effective problem-based learning environments teach us that solving real-world problems requires scaffolding, i.e., help from facilitators, knowledgeable experts, and the learning environment [12, 18]. The goals of scaffolding are to enable students to carry out a reasoning process or

achieve a goal that they would not be able to do without help, and to facilitate learning to achieve the goal without support. The scaffolding of different skills can be provided through software, by appropriately utilizing multimedia and tools such as collaboration software, simulation and visualization programs, decision-support systems and smart case-libraries.

(5) A shared electronic workspace that seamlessly integrates a full variety of functionalities for the above will enhance learning. This workspace will tie together tools that students will use while solving problems, collaborating, and perusing multiple cases. It is also an ideal vehicle for providing adaptive software-realized scaffolding of various skills. Finally, it will encourage both synchronous and asynchronous collaborative work among students. Such an integrated yet flexible computer-based learning environment that the students use as a "professional workspace" is a central component of our approach.

We want to situate classroom learning in information-rich contexts that afford opportunities for problem formulation, exploration, and discovery. Students will work on problems for extended periods of time, reflecting and articulating on both the process and the product. Case libraries will provide them with both relevant data and specific solution strategies in the domain of instruction, all within the context of complex and realistic real-world problems. The problems students have to solve and the cases that are made available to them serve as anchors for learning. Collaborative, reflective and articulative activities, aided by the tools and cases provided by the computer-based learning environment, should improve the students' knowledge, problem solving skills, and self-directed learning skills. Cases, being rich knowledge structures that explicate both conceptual and strategic knowledge, will allow the students to master concepts, principles and strategies in the course of attempting to solve problems. The collaborative nature of student activities should facilitate the construction of new knowledge since it encourages articulation and intra-group communication. Our approach is designed in particular to address the following three issues.

Cognitive Flexibility and Transfer. Consideration of a single case leads to inflexibility of the acquired knowledge and strategies [22]. Rather than having students focus on a single case, our intention is to have students revisit ideas from multiple cases both through the design problems that students work on and the design cases in the case libraries. We believe that by having students analyze multiple cases, and by having them reflect on how these cases are similar and different to the problems they are solving, more flexible knowledge should be constructed. The cognitive flexibility theory [20] supports this prediction.

Collaboration. Collaboration is a key piece of our approach. Research on collaborative learning shows

that learning while solving problems in groups facilitates the learning of articulation skills, makes learning more effective for all group members, and allows students to successfully tackle problems more complex than any one group member could individually solve [3, 14, 17]. Moreover, the collaborative discussion that occurs is important for student learning because it activates prior knowledge, thus facilitating the processing of new information [2, 19]. On the other hand, Blumenfeld et al. [1] suggest that students may have more motivation to learn but make less use of learning and metacognitive strategies. In addition students may not have the skills to benefit from collaborative work. Therefore it is important to help students to collaborate well together in order to make collaborative learning work well. Aspects of our approach - the division of the student body into small groups, the complexity of the design problems that the groups will tackle, and the use of collaboration software to scaffold communication and cooperative work - are all intended to overcome these limitations and enhance the benefits of group-oriented learning.

Reflective Articulation. Two important aspects of our approach are articulation and reflection. There are several forms of reflective articulation including generating analogies [10], predicting outcomes of events or processes [21], developing questions about the learning materials [10], and self-explanations [4]. Studies suggest that reflective articulation can enhance retention, elucidate the coherence of current understanding of the problem being solved, develop self-directed learning skills, and provide a mechanism for abstracting knowledge from the content in which it was learned, thus facilitating transfer. It is very important to provide two levels of articulation - individual and group - in a collaborative learning environment. Also, it is not just articulation by itself that is important, but it is the specific kinds of articulations that engenders reflection - *reflective articulations* - that lead to enhanced understanding. The goal of reflection is to analyze and evaluate one's knowledge, learning and problem solving strategies. Several researchers have demonstrated the importance of articulation and reflection in learning. Pirolli and Recker [15] suggest that reflection on problem solutions that focuses on understanding the abstract relationships between problems is related to improved learning. Lin [13] has found that reflection on problem-solving processes leads to enhanced transfer and that technology can be used to scaffold appropriate kinds of reflections. One way that reflection can be enhanced is through the articulation of meta-cognitive knowledge and skills that typically occurs in collaborative discourse.

3. Computer Support for Collaborative Learning in McBAGEL

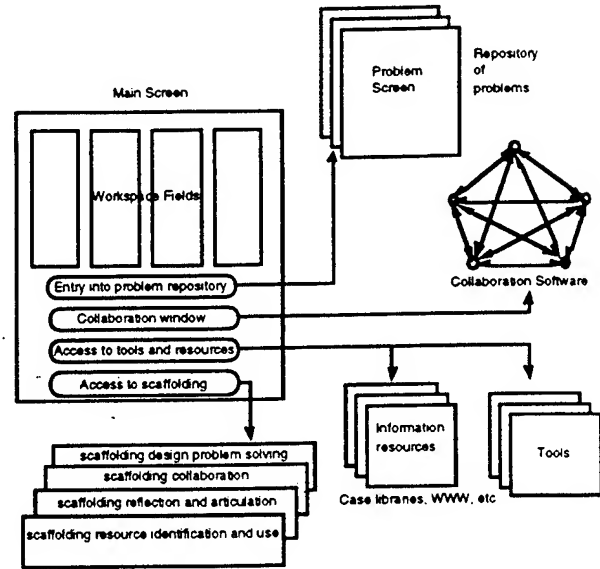


Figure 1. Software Architecture

Figure 1 is a schematic diagram of the architecture of a software environment that we are developing to complement the McBAGEL approach in classrooms. This environment provides an external memory for keeping track of problem specifications, important facts and constraints, ideas about how to deal with the specifications, and learning requirements. The main screen provides several fields for keeping track of multiple sources of information, design alternatives, and further actions to be taken. Space is provided to record the facts and constraints that are important, to record ideas about how to deal with the specifications, and to keep track of what else needs to be learned, what information needs to be collected, and what actions need to be taken. Together, these windows allow the student to see where s/he is now, where s/he has been, and where s/he is going. This screen can be used as an individual workspace or as a shared workspace for the group. The main screen also provides access to other resources and tools that students need to solve the design problems: case libraries and other information resources; tools for simulation, visualization, decision making etc.; a tool for inter- and intra-group communication, collaboration, and multimedia document sharing; and a set of basic tools such as document processing programs, drawing/painting programs and spreadsheets.

The problem screen provides easy access to the evolving problem description. This screen begins with a minimal description of the design problem presented to the students. Details emerge as they inquire about additional information about constraints, material resources and functional issues regarding the design.

The collaboration window allows students to enter into a collaboration environment that provides much more than mere communication facilities. It will

provide an ability to enter into structured discussions on different topics pertaining to the class and the problem at hand as well as to share multimedia resources with other members of the group and class. A user will be able to browse through past and ongoing discussions which are presented in a structured format to allow easy topic-based, time-based or author-based browsing, and to contribute to those discussions by constructing and sending different types of messages. This collaboration facility will be made available to not only students, but also to teachers. It will provide teachers with a means to collaborate in conducting a course and to share experiences and learn from each other. It can also be a vehicle for student assessment based on their collaborative interactions.

In addition to providing a work environment, this system makes available scaffolding to help novices with design, collaboration and reflection. Design scaffolding will vary as a function of the design stage students are working on. For example when the students are working on problem formulation, the software will provide coaching to help them understand what is involved in this stage: e.g., identifying the problem, formulating the problem, partitioning/decomposing the problem, and framing the problem. The collaboration software will provide procedural facilitation to aid in the development of collaboration skills. Reflection will be facilitated through the articulation that occurs during collaborative problem solving and learning activities.

In summary, this environment will provide means to organize and manage projects from the students' perspective (e.g., the main screen provides for explicitly listing organizational and learning issues) and the teachers' perspective (e.g., tracking student progress and keeping records of student work). In addition, we envision that the environment will be used for research purposes (e.g., archiving data such as the inter- and intra-group communications and resource sharing that took place during a course for later assessment, collecting data to be used for student/group modeling in order to devise better course- and student-specific on-line scaffolding and coaching methods, etc.). An initial prototype of this environment has been developed with Hypercard on the Macintosh platform, but it has not yet been tested in a classroom. Borrowing from the metaphor of the white board workspace of problem-based learning found in medical schools, this prototype provides an electronic workspace that is split into four regions. It also allows easy access to other tools and resources. Figures 2 and 3 show the workspace and problem screens of this prototype.

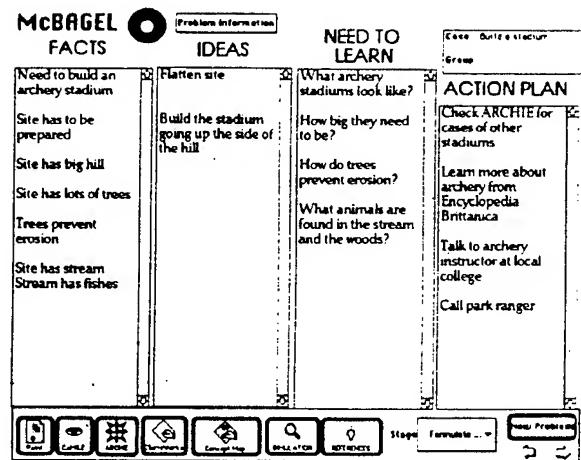


Figure 2. McBAGEL Workspace Screen

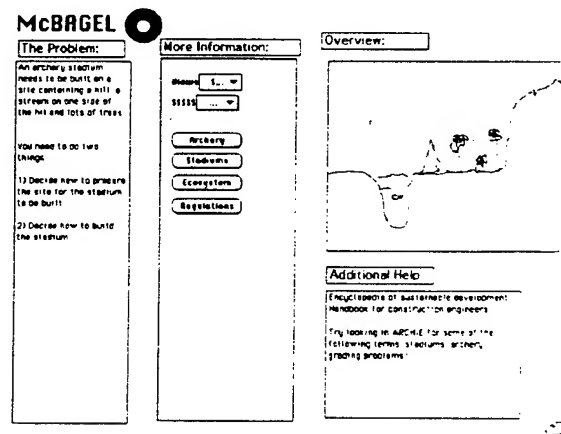


Figure 3. McBAGEL Problem Screen

Here is a brief scenario to illustrate how we imagine the students will use this environment. Students, who will be working in small groups, enter the environment at the main window shown above, which represents their shared electronic workspace. They are provided with relevant information on the design problem they need to solve via the button "new problem". In this case, it is to design an archery stadium for the Olympics. As students are initially formulating and understanding the problem, they will be encouraged to identify data relevant to the problem from the information they have been provided with, and to articulate this by recording those in the "facts" space. Similarly, as they consider alternative solutions, they will make use of the "ideas" space. The problem-based learning methodology that this environment embodies explicitly prepares students for self-directed learning by requiring them to identify their knowledge deficiencies in the "need to learn" space and the actions they plan to take to remedy those deficiencies in the "action plan" space. Several buttons are found on the bottom of the screen that provide access to different tools that they will need to solve the problem.

"Stage" is a pull-down menu which acts as a gateway to various kinds of software-realized scaffolding tailored to different stages of problem solving.

While the structure of this environment is still evolving, some of its components have already been designed, implemented and individually fielded in classrooms. In the following two sections we elaborate on these implemented components and describe future directions for our research.

4. Implemented Components

- *Case Libraries*: Research on case-based reasoning [11] provides guidelines for indexing and making available resources needed while problems are being solved, especially case materials. Case libraries organize cases in ways that make it easy to access their most interesting parts, understand their implications, and recognize the range of problems needing solving and the range of solution methods available. Case studies are structured in terms of overviews, problems, stories, and responses. Each story discusses some problem that arose in designing some artifact, the way that the problem was addressed, and the outcomes that resulted. To make it easy for users to extract from stories their important points, stories are presented with illustrative graphics, and several kinds of contextual information is associated with each story. Students can examine the full artifact that some story is associated with, can see a general description of the problem the story addresses, a general description of the kind of solution it provides, and can ask to follow links to other stories that illustrate a similar problem or solution. The stories help students discover which issues they should be considering during design and help them to anticipate the results of carrying out their proposed designs. We have developed a number of case libraries in support of design problem solving.

- *Case Library Authoring Tool*: DesignMUSE [5] is a case library authoring tool that has been developed to allow easy construction of case libraries. During the 1995 Winter Quarter it was used to create a library of environmental cases for use in our sustainable technology classes. Thus, while existing case libraries act as intelligent information resources, this authoring tool will allow students to construct their own case libraries to record the design problems that they solve. Both the authoring tool and the case libraries are built on Common Lisp for Macintoshes.

- *CaMILE*: Our collaboration software CaMILE [8], based in principle on CSILE [18], integrates information-gathering tools, communication tools, and applications into a collaborative environment. CaMILE provides a discussion environment into which the full range of text, graphics, spreadsheets, video, and so on that reside locally or on the Internet can be incorporated. It is designed to meet two goals. First, it serves as a collaboration and information indexing tool. Discussions are structured and annotated with

links to material anywhere across the network. Second, it serves as a design support tool. Discussions about design problems can be annotated with links to actual ongoing designs. The discussion trace can then serve as a design rationale and a case study of a design. It allows students to collaborate in learning and problem solving by providing a facility for structured inter-group and intra-group communications that are archived, and by providing a way to share multimedia documents easily among collaborators. Like CSILE, CaMILE scaffolds collaboration through procedural facilitation. While electronic mail merely allows team members to share ideas, CaMILE helps them to organize their ideas into coherent arguments, relate their ideas to one another, and use resources across the network to support their arguments. CaMILE was built with Hypercard on Macintosh computers.

- *Exploratory Simulations*: We have developed a range of exploratory simulations [16] that enable students to learn through simulated experience. Key to these simulations are tight integration with real world problems and activities, and flexible specification of simulation choices to allow for creative and sophisticated simulation problem solution. These simulations have been constructed using the Smalltalk language.

5. Future Directions

Learning from case libraries: As students are solving problems, several kinds of resources are needed to help them. Clearly, they need access to documentation, of the kind found in books and encyclopedias. But another significant but often overlooked resource is codified prior experience: e.g., cases that describe solutions to similar problems. Our approach to supporting learning from prior experience is to make on-line case libraries available from within the software environment. Cases help with understanding a problem better, suggesting solutions and parts of solutions, and evaluating proposed solutions, thereby helping a student to know where to focus his/her attention. Our research on case libraries will proceed in two directions. One is generating content: creating the kinds of cases with which to populate these libraries in order to have maximum impact on learning. The other concerns issues of information organization, presentation and navigation. How can cases be organized and presented in ways that make it easy to access their most interesting parts, understand their implications, and recognize the range of problems and the range of solutions available? While the existing case libraries provide one answer to this question (another, for example, is provided by [9]), we are currently revisiting this issue from the perspective of students, who are novice practitioners. From this perspective we believe that additional capabilities such as access to definitions of the terms used by experts, access to explanations of what experts find it

appropriate to focus on, guidance in choosing what to focus on next, and allowing students to extend the libraries (or create new ones) are also required.

Supporting collaborative problem solving and learning: Support for group communication and sharing will be provided by facilitating collaborative work through the software environment. CaMILE was used during the past two quarters in a junior-level design foundations course, taught in mechanical engineering (ME 3110; Creative Decisions in Design). We have collected data on the system's usage and its effects, and are in the process of analyzing this data. A World Wide Web version, WebCaMILE, is also under development. We plan to link case libraries and WebCaMILE so that students engaged in a design activity might use WebCaMILE to discuss and exchange case-study materials. Cases provide the kinds of information that a student might point to as justification for some argument presented to others, as a potential alternative to a design decision, or as a rebuttal to someone else's design decision. Tailoring CaMILE's procedural facilitation to reflect more closely the content and nature of the problems students will be solving and investigating new ways of scaffolding collaboration are other topics of ongoing research.

Software-realized scaffolding: Of particular importance in making this integrated software environment work for students is providing software-realized scaffolding to support student use of the environment for learning. We have identified several specific areas in which we can provide facilitation.

- Scaffolding collaborative design and problem-solving: Our environment will provide scaffolding for design and problem-solving using several techniques:

- By structuring the kinds of entries which can be made in a group discussion, e.g., new theories or ideas, alternatives, comments, rebuttals, and questions. When a student chooses one of these kinds of entries, an editor opens for their comments and a prompting window opens with suggestions for useful entries to make, e.g., for a rebuttal, suggestions might include "The strengths of this idea are..." and "But the key weakness is...". This scaffolding guides the discussion in useful directions defining the kinds of entries to be made, asking students to choose one before entering an item into the discussion, and suggesting appropriate things to say.

- By providing agents to actively review student work and suggest better ways to design and solve problems. For example, agents may identify where connections might be made between efforts, where additional resources exist that might aid an effort, and where efforts may be going astray [6].

- By providing menus of glossaries of relevant vocabulary and their definitions.

- By providing means of visualization and making explicit the design process.

- Scaffolding reflection and learning: We want to support two kinds of reflection in the environment because we believe that reflection can significantly facilitate learning.

- Reflection-in-action: The students' articulations in the discussion, the declaration of item type, and the linking of cases to discussion are all forms of reflection-in-action. These are kinds of reflection which are integral to the design process and which support both the execution of a good design process and the learning about that process. Reflection-in-action helps to make strategies explicit and learnable, develops an expanded repertoire of strategies, and improves student understanding and control of the design process.

- Reflection-as-summary: Student summarization at the end of a design process is an important learning activity for students and an important resource for future groups of students. Our plan is for students to summarize their group design projects such that summaries from one class become cases in the library for the next class. Thus, students summarize not just for their own benefit but to help a future audience.

- Scaffolding resource identification and use: Case libraries support student exploration by providing multiple indices into cases. Students might begin by looking at one case of interest and then explore related cases by a number of different dimensions, or begin by browsing all cases related to a problem. Students can gain perspective on what problems they are facing, what the parameters of the problems are, and how these parameters are explored in the cases in the library from case overviews. We want case libraries to provide support for all these kinds of searching and browsing, but coupled with support that helps in applying the found information to the task at hand (e.g., linking cases that highlight an important alternative solution to the discussion on that alternative). In addition, we envision the use of visualization tools to aid in resource identification and use.

Integration: As many of the critical components of the software environment are being implemented and used in classrooms, the most significant task ahead of us is integrating the different pieces into a single environment. This integrated environment supporting the McBAGEL approach has to play several roles: facilitation of design problem solving and its constituent components, facilitation of learning, access to resources, and access to teachers and fellow learners. The software environment has to serve as both an electronic workspace and a learning environment providing help with a variety of intellectual activities as students collaborate on design projects. We see a need for this environment to promote reflection and summarization as well. Software-guided reflection is particularly important in facilitating skill transfer between different problem

domains. The construction of such an environment on Macintosh computers is currently underway.

Assessment: The next step, slated to begin in Fall 1995, is to use and assess both the approach and the concomitant software environment in a series of design courses at Georgia Tech. We will use assessments to determine what kind of learning has occurred and how well students apply what they have learned. The goals of learning involve not merely acquiring a set of static facts to be recalled on a test but rather involve constructing a coherent understanding of a domain that can be flexibly transferred to new situations. The extent to which learning can be used in new situations (i.e., transfer) allows assessment of how flexibly the students have learned the content and are able to apply it to complex problems. Students' learning will be evaluated on mastery, near-transfer, and far-transfer problem-solving. Cognitive research suggests that because problem-based instruction is geared towards complex curricular objectives, assessments need to include open-ended questions in which students explain what approaches they have to a problem and its solution [7]. A variety of methods will be used to collect this data including interviews and paper-and-pencil short answer tests. This allows measurement of the products and processes of the students' learning. Some authentic performance assessments will also be devised. Students' presentations will be assessed to examine how they define the problems and justify their solutions as well as the quality of their solutions. Because transfer is not an all-or-none phenomenon, different types of transfer will be assessed and measures will be developed that assess this. We will use measures of knowledge, skills, planning, and qualitative understanding as students are asked to justify their solutions. This will assess the flexibility of the knowledge that the students construct. For example, because of the emphasis on problem solving, we would expect increased integration of the content they are learning into their problem-solving on transfer problems. Because students are using the collaborative environment and gaining experience and feedback in articulating their plans for problem-solving, we expect improvement in the students' planning skills as well.

6. Conclusions

Collaborative learning environments have the potential for helping students to construct usable knowledge and to learn strategies that prepare them for a lifetime of learning. To afford generative learning, such environments need to contain rich sources of information. In addition, opportunities for student collaboration, articulation and reflection must be provided to help students think deeply about the problems they are working on and to learn to go beyond the given problems. The McBAGEL approach is designed to meet these requirements. Providing

computational support to this approach requires the design of a software architecture that integrates multiple tools and information resources with a structured electronic workspace. This paper describes our efforts on developing the theoretical and practical aspects of such an architecture. The focus of our current research is on refining and testing the components further, and on fully implementing the integrated environment. Future research will focus on deploying it in classrooms and conducting assessments of its impact on student learning.

Acknowledgments

Research reported here has been supported by ARPA under contract N00014-91-J-4092 monitored by ONR, ONR under contract N00014-92-J-1234, and by the Woodruff Foundation's support of the EduTech Institute.

References

1. Blumenfeld, P., Soloway, E., Marx, R., Krajcik, J., Guzdial, M., and Palincsar, A. 1991. Motivation project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist*, Vol. 26, Nos. 3&4, pp. 369-398.
2. Bransford, J. D., and Johnson, M.K. 1972. Contextual prerequisites for understanding: Some investigations of comprehension and recall. *Journal of Verbal Learning and Verbal Behavior*, Vol. 11, pp. 717-726.
3. Brown, A. L., and Palincsar, A. S. 1989. Guided, cooperative learning and individual knowledge acquisition. In *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, L. B. Resnick (Ed.). Erlbaum, Hillsdale, NJ, pp. 393-451.
4. Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., and Glaser, R. 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, Vol. 13, pp. 145-182.
5. Domeshek, E.A., and Kolodner, J.L. 1992. A case-based design aid for architecture. In *Proceedings of the Second International Conference on Artificial Intelligence and Design*.
6. Fischer, G., Lemke, A. C., Mastaglio, T., and Morch, A. I. 1991. The role of critiquing in cooperative problem solving. *ACM Transactions on Information Systems*, Vol. 9, No. 3, pp. 123-151.
7. Glaser, R., and Silver, E. 1994. Assessment, testing, and instruction: Retrospect and prospect. In *Review of Research in Education*. Vol. 20. Darling-Hammond (Ed.). American Educational

- Research Association, Washington D. C., pp. 393-419.
8. Guzdial, M., Rappin, N., and Carlson, D. 1995. Collaborative and multimedia interactive learning environment for engineering education. In *Proceedings of the ACM Symposium on Applied Computing*, pp. 5-9.
9. Hsi, S., and Agogino, A. M. 1994. The impact and instructional benefit of using multimedia case studies to teach engineering design. *Journal of Educational Multimedia and Hypermedia*, Vol. 3, Nos. 3/4, pp. 351-376.
10. King, A. 1992. Comparison of self-questioning, summarizing, and notetaking-review as strategies for learning from lectures. *American Educational Research Journal*, Vol. 29, pp. 303-323.
11. Kolodner, J. 1993. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA.
12. Koschmann, T. D., Myers, A. C., Feltovich, P. J., and Barrows, H. S. 1994. Using technology to assist in realizing effective learning and instruction: A principled approach to the use of computers in collaborative learning. *Journal of the Learning Sciences*, Vol. 3, pp. 225-262.
13. Lin, X. 1994. Far-transfer problem-solving in hypermedia environment: The role of self-regulated learning processes. Paper presented at the American Educational Research Association Annual Meeting, New Orleans.
14. Pea, R. D. 1994. Practices of distributed intelligence and designs for education. In *Distributed Cognitions*, G. Salomon and D. Perkins (Eds.). Cambridge University Press.
15. Pirolli, P., and Recker, M. 1994. Learning strategies and transfer in the domain of programming. *Cognition and Instruction*, Vol. 12, pp. 235-275.
16. Rappin, N., Guzdial, M., Ludovice, P., and Realff, M. 1995. DEVICE: Dynamic environment for visualizations in chemical engineering. In *Proceedings of the AI-Education Conference*, (In Press).
17. Riel, M. 1993. Global discourse via electronic networking. Paper presented at the Annual Meeting of the American Educational Research Association, Atlanta, GA.
18. Scardamalia, M., Bereiter, C., McLean, R. S., Swallow, J., and Woodruff, E. 1989. Computer-Supported Intentional Learning Environments. *Journal of Educational Computing Research*, Vol. 5, pp. 51-68.
19. Schmidt, H. G., DeGrave, W. S., DeVolder, M. L., Moust, J. H. C., and Patel, V. L. 1989. Explanatory models in the processing of science text: The role of prior knowledge activation through small group discussion, *Journal of Educational Psychology*, Vol. 81, pp. 610-619.
20. Spiro, R. J., Coulson, R. L., Feltovich, P. J., and Anderson, D. K. 1988. Cognitive flexibility: Advanced knowledge acquisition in ill-structured domains. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Erlbaum, Hillsdale, NJ, pp. 375-383.
21. White, B. 1984. Designing computer activities to help physics student understand Newton's laws of motion. *Cognition and Instruction*, Vol. 1, pp. 69-108.
22. Williams, S. M., Bransford, J. D., Vye, N. J., Goldman, S. R., and Carlson, K. 1993. Positive and negative effects of specific knowledge on mathematical problem-solving. Paper presented at the American Educational Research Association Annual meeting, Atlanta GA.

Authors Addresses

N. Hari Narayanan, Cindy E. Hmelo, Valery Petrushin, Wendy C. Newstetter, Mark Guzdial and Janet L. Kolodner. EduTech Institute, College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280. {narayan, ceh, petr, wendy, guzdial, jlk}@cc.gatech.edu.

A Multiple-Case-Based Approach to Generative Environments for Learning

Cindy Hmelo, N. Hari Narayanan, Wendy C. Newstetter, and Janet L. Kolodner

Edutech Institute
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
[ceh, narayan, wendy, jlk]@cc.gatech.edu

Abstract

In this paper we present a vision of learning through solving generative problems - problems that promote open-ended inquiry and have multiple solutions. This vision stems from a novel and evolving approach to learning that we are developing at the EduTech Institute. This approach is based on the following premises: that learning is facilitated by generative problem solving, collaborative work and use of multiple cases; that learning and skill acquisition need to be, and can be, scaffolded through software; and that a computer environment which integrates a shared and structured electronic workspace with a full variety of functionalities can effectively support all of the above. We describe this approach and the architecture of the corresponding computer environment. This environment is designed to serve three critical functions: provide a shared workspace for students, facilitate inter- and intra-group collaborative work, and make available the tools and resources that students need for problem solving and learning. The software components of the environment that have already been implemented are described. Finally, issues and questions driving current research are outlined.

1. Introduction

Too often students learn bits of knowledge that they are unable to assemble and apply in productive ways. Modern theories of learning suggest that for learning to occur and to be useful, the learner needs to be *actively involved* in meaningful contexts (e.g., Brown, Collins, & Duguid, 1989). Situating learning in rich problems or cases is one way of accomplishing this. Case-based learning (CBL) means that, as students solve a problem or study a case, they construct knowledge at the same time (Williams, 1993). In this paper we present a multidisciplinary approach to structuring learning environments within the framework of CBL, called Multiple Case-Based Approach to Generative Environments for Learning (McBAGEL). At the EduTech Institute, we are developing McBAGEL for use in a variety of engineering and middle school courses that involve ill-structured problem-solving. Several factors characterize our approach: (1) the use of design problems to promote generative learning; (2) using design cases provided by computer-based case libraries as resources to aid problem solving and foster deep understanding; (3) scaffolding the acquisition of knowledge and skills through a combination of software support and careful orchestration of classroom activities; (4) providing feedback in a variety of forms; (5) emphasizing group-oriented collaborative work in and out of the classroom; (6) encouraging reflective articulation; (7) making available to students an electronic sharable workspace integrated with tools and resources as the medium for work; and (8) redefining the role of the teacher as a facilitator of learning. Because we are dealing with design-oriented problems, there is a great deal of complexity to be managed. The pedagogy and software support is geared towards supporting learners in managing the complexity and emphasizing *learning as well as doing*.

McBAGEL comprises both a pedagogical approach *and* a software architecture designed to complement and support effective classroom practice of this approach. We are engaged in implementing this approach both at the college level (engineering undergraduates) and at the middle school level. We are currently developing design problems and case libraries for engineering design and middle school math and science curricula, as well as prototyping the computer-based electronic workspace. In this paper, we discuss the theoretical foundations of our approach, the architecture of the electronic workspace, and its software components.

2. Foundations of Case-Based Learning: A Multidisciplinary Perspective

Case-based learning (CBL) uses cases or problems as stimuli for learning and foci for organizing what has been learned (Barrows, 1994). This means that students learn while solving a problem or studying a solved case. Characteristics of CBL environments include:

- Emphasis on collaborative student-centered learning.
- Emphasis on learning knowledge and strategies.
- Students solve the problem and make their thinking visible by justifying their solution process using theory, causal models, or other appropriate evidence.
- Problems are ill-structured, the data are embedded in the problem itself, and are often emergent

as the problem is explored.

McBAGEL, our approach to CBL, draws from research in Cognitive Psychology, Education, and Case-based Reasoning. Relevant perspectives from these areas are discussed below.

Cognitive Psychology. There is both cognitive theory and evidence supporting the benefits of case-based approaches to learning (Norman & Schmidt, 1992). The cognitive psychology literature provides support for CBL in three roles. First, the acquisition of factual knowledge in the context in which it will later be used should enhance later retrieval (Adams, et. al., 1988; Perfetto et. al., 1983). Second, the mastery of concepts, principles, and strategies while attempting to solve a problem should promote transfer to new problems (Catrambone & Holyoak, 1989; Needham & Begg, 1991). Needham and Begg (1991) have demonstrated the importance of feedback in facilitating transfer to new problems, suggesting that this should be an important feature of CBL. Third, the acquisition of prior examples that can be used for problem solution on the basis of similarity is valuable. However, for exposure to earlier problems to be helpful the learner needs to see multiple examples and reflect on them (Brooks et. al., 1991; Chi et al., 1989; Gick & Holyoak, 1983). Moreover, the collaborative discussion that occurs in CBL is important for student learning because it activates prior knowledge, thus facilitating the processing of new information (Bransford & Johnson, 1972; Dooling & Lachman, 1971; Schmidt et. al., 1989). In CBL students are encouraged to think about problems with the underlying principles in mind rather than just collecting sets of features. Chi et al. (1989) note that during self-explanations cases get connected to domain principles and the learner begins to understand how knowledge can be applied to solving problems, leading to more flexible knowledge.

Cognitive Flexibility Theory (CFT) provides evidence that suggests case-based learning methods are important for learning in complex, ill-structured domains (Spiro, Feltovich, Coulson, & Anderson, 1988). A domain is ill-structured when cases are multidimensional and irregularly related to other cases and to the underlying causal models (if such models exist). By this definition, design is ill-structured. This means that, although the underlying science and mathematics used in a design problem may be well-structured, "there is variability from case to case regarding which conceptual elements are relevant and in what pattern of combination" (Spiro et al., 1988, p.379). CFT also suggests that knowledge will be learned flexibly if students have the opportunity to explore the problem from multiple points of view. The implication of this is that for cases and concepts to be understood and useful as problem-solving tools, they need to be visited from a variety of different experiences.

Education. Research in education provides us with two examples of case-based learning (e.g., Barrows, 1985, 1994; CTGV, 1990, 1993). Anchored instruction situates learning and problem-solving in rich contexts with meaningful goals. Students solve complex, realistic mathematics problems using a video-based story to present the problem. For example, *The Adventures of Jasper Woodbury* (CTGV, 1990;1993) video series depict realistic situations in which mathematical problem solving is required. To solve these problems, the students must formulate their problem-solving goals and plan how they will achieve those goals. They must identify the information relevant to solving the problem as is required in complex real-world problem-solving. Students initially brainstorm to determine the goals and data that would be needed to solve the problem. Then, students break into small groups to work on the subproblems or subgoals that they generate. Groups present the results of their problem solving to the class and get feedback from the class and teacher. While working in small groups, students can review the video to gather data and ask the teacher questions. They learn about mathematics in the course of solving the problem.

Anchored instruction affords opportunities for problem finding, exploration, and discovery (CTGV, 1990;1993). The stories provide for generative learning because the learners must complete the story through their problem-solving efforts. An important feature of the stories is that the data are embedded within them. The problem-solvers must learn to differentiate between relevant and irrelevant data. The problems used are much more complex than, for example, typical math word problems on the premise that children cannot learn to deal with complexity unless they have had the chance to experience it.

Problem-based learning (PBL) is a CBL methodology developed originally for medical students (Barrows, 1985). Students in PBL learn through solving real patient problems. The PBL approach begins with the teacher presenting a problem to students by showing materials or facilitating a discussion that "brings the problem home" to students. The students are presented with an initial scenario that specifies a problem to be solved and the particular product or performance that they must achieve. As they try to better understand the problem, they question the facilitator to get additional case-information. At several points in their problem solving, the students pause to reflect on the data they have collected so far, to generate questions about the data, and to hypothesize about underlying causal mechanisms or solutions for the problems. Students identify issues that they do not understand and need to learn more about. After considering the problem with their naive knowledge, students research the learning issues they have identified, by going to the library, consulting an expert, interacting with a computer program, or in whatever way is appropriate. They later share what they've learned, attempting to apply their new knowledge in solving the problem. They might refine hypotheses or options or generate new ones based on their new knowledge. The cycle of deliberating together and separating to find out new things continues until the problem is solved. When finished, the students present their product and then intentionally reflect on what they have learned. In addition, they assess their own and other group members' contributions to the group's learning and collaboration.

PBL has several goals. First, in order to make basic science knowledge available in a clinical context, students learn science

by having it embedded in a patient problem. This should help students better integrate scientific and clinical knowledge, thereby improving their access to basic science information when they need it in the clinical context. A second goal is to facilitate the development of clinical reasoning skills. A third goal is to facilitate the development of skills in self-directed learning and self-assessment. An explicit objective of PBL is to increase students' sensitivity to their personal learning needs and their skill at locating and using appropriate information resources. Finally, PBL is expected to enhance student motivation. PBL provides an appropriate context for learning because all of the content is learned in the context of a problem which should enhance recall of this information when it is needed. Norman and Schmidt (1992) report several direct tests of the effect of PBL on recall. These studies indicate that PBL students' initial learning is not as good as students in a conventional curriculum but that their long term retention is superior. Research has shown that students in a PBL environment are more likely to use science in their problem-solving than traditional students (Hmelo, 1994).

Case-Based Reasoning. Case-Based Reasoning (CBR) provides insights that complement the CBL methodologies. It provides suggestions for sequencing problems to form a curriculum as well as the kind of reflection that is needed (Kolodner, 1995). CBR means reasoning based on previous experiences (Schank, 1982). It might mean solving a new problem by adapting an old solution or merging pieces of several old solutions, interpreting a new situation in light of old similar situations, or projecting the effects of a new situation by examining the effects of a similar old situation. In short, case-based reasoning means using the lessons learned in old situations to understand or navigate new ones. The basic premise underlying case-based reasoning is the preference to reason using the most specific and most cohesive applicable knowledge available. Inferences made using cohesive knowledge structures, i.e., those that tie together several aspects of a situation, are relatively efficient. Cases, which describe situations, are both specific and cohesive. In addition, they record what is possible, providing a reasoner with more probability of moving forward in a workable way than is provided by knowledge that is merely plausible.

Research in CBR (Kolodner 1993) indicates several ways of enhancing CBL. It suggests that knowledge will be more accessible, flexible, deeply learned, and accurate if learners have the opportunity to encounter (first-hand or by report) multiple situations in which the knowledge is used and multiple ways in which similar situations are addressed, and if students have the opportunity to reuse and try out knowledge gained through experience. Further, CBR suggests that the experiences of others be made available to students to model successful reasoning, to help students get started, to point the way to issues that need to be addressed, and to fill in where the full range of real experience is impossible or infeasible. It also suggests that problems and products should afford failure of expectations and that the environment needs to afford the kinds of feedback that will allow successful analysis of failures. Finally, it suggests that reflection should focus on anticipating the uses of lessons learned through each experience and that facilitation should refer back to previous experiences of the students to help them notice similarities and abstract from the range of problems they've solved.

3. Synthesis

When we put all that we have learned from research in cognitive psychology, education, and case-based reasoning together, four principles emerge:

1. Learning is enhanced by generative problem solving. Learning is most effective when it occurs through generative activities associated with solving problems (e.g., identifying and formulating the problem, generating alternatives, evaluating, decision making, reflecting, and articulating). Design, by its very nature, is a generative activity. Therefore, design-oriented problems are particularly effective for technical domains like engineering as well as providing effective contexts for math and science learning.

2. Collaborative work promotes knowledge building. Students are expected to solve problems and do assignments in groups. Group-oriented work, in and out of the classroom, is important both in facilitating learning and in preparing students for today's multidisciplinary team-oriented workplaces. As students work in collaborative groups, they are forced to articulate and reflect upon their thinking, leading to an appreciation of the importance of distributed cognition (Pea, 1993) as well as enhancing learning and subsequent transfer (Brown & Palincsar, 1989). Collaborative work allows students to successfully tackle problems more complex than what any one group member could do alone. The collaborative discussion that occurs is important for student learning because it activates prior knowledge, thus facilitating the processing of new information (Bransford & Johnson, 1972; Schmidt et al., 1989). The dialogue among group members enriches and broadens the groups' problem solving process (Vye, Goldman, Means, Voss, Hmelo, & Williams, 1995). On the other hand, Blumenfeld et al. (1991) suggest that students may have more motivation to learn but make less use of learning and metacognitive strategies. In addition students may not have the skills to benefit from collaborative work. Therefore it is important to help students to collaborate well together in order to make collaborative learning work well.

3. Articulation and reflection are central to learning. Several researchers have demonstrated the importance of articulation and reflection to learning. The goal of reflection is to analyze and evaluate one's knowledge, learning and problem solving strategies. Pirolli and Recker (1994) suggest that reflection on problem solutions that focuses on understanding the abstract relationships between problems is related to improved learning. Lin (1994) has found that reflection on problem-solving processes leads to enhanced transfer and that technology can be used to scaffold appropriate kinds of reflections. One way that

reflection can be enhanced is through the articulation of metacognitive knowledge and skills that typically occurs in collaborative discourse.

4. Access to multiple cases will facilitate flexible learning. Providing students with access to multiple cases that contain rich descriptions of specific situations can significantly enhance learning and transfer. The use of multiple cases as resources for learners' problem solving both facilitates learning new knowledge, and supports the skill of transferring previous solutions to the current problem. It is expected that by revisiting concepts and skills through numerous cases, flexible transfer will be promoted (Spiro, Coulsen, Feltovich, & Anderson, 1988). Consideration of a single case leads to inflexibility of the acquired knowledge and strategies (Williams, Bransford, Vye, Goldman, & Carlson, 1993). Therefore, it is important to provide students with access to multiple cases that are relevant to their activities during various stages of design problem solving. Computer-based case libraries can provide students with not only such access but also means of flexibly navigating among cases and parts of cases. Such libraries are therefore information-rich and powerful resources for learning. Students will use case libraries in two significant ways. One is by searching for, analyzing, comparing and contrasting cases that are similar to the problems they are solving. The other is by constructing cases from domain knowledge and from their problem solving experiences and incorporating these into the case libraries.

4. McBAGEL

Our approach, called McBAGEL, derives from these principles. McBAGEL proposes to situate classroom learning in information-rich contexts that afford opportunities for problem formulating, exploration, and discovery. Students work on problems for extended periods of time, reflecting and articulating on both the process and the product. Case libraries provide them with both relevant data and specific solution strategies in the domain of instruction, all within the context of complex and realistic problems. The problems students have to solve and the cases that are made available to them serve as anchors for learning. Collaborative, reflective and articulative activities, aided by the tools and cases provided by the computer-based learning environment, are expected to improve the students' knowledge, problem solving skills, and self-directed learning skills. Cases, being rich knowledge structures that explicate both conceptual and strategic knowledge, will allow the students to master concepts, principles and strategies in the course of attempting to solve problems. Students will revisit ideas from multiple cases both through the sequence of design problems that they work on and the design cases they access in the case libraries. We believe that by having students analyze multiple cases, and by having them reflect on how these cases are similar and different from the problems they are solving, more flexible knowledge should be constructed. Cognitive flexibility theory supports this prediction (Spiro et al., 1988). The collaborative nature of student activities should facilitate the construction of new knowledge since it encourages articulation and intra-group communication.

We take a holistic approach that includes development of curriculum-appropriate design problems, scaffolding the complexity, software development and teacher involvement. Our approach is distinguished by the following features:

1. Scaffolding. Solving real-world problems is hard. Others' experiences on implementing effective problem-based learning environments teach us that solving complex problems requires scaffolding, i.e., help from facilitators, knowledgeable experts, and the learning environment, to help students manage the complexity of problem-solving and to promote learning (Koschman, Myers, Feltovich, & Barrows, 1994; Scardamalia, Bereiter, McLean, & Woodruff, 1989). The goals of scaffolding are to enable students to carry out a reasoning process or achieve a goal that they would not be able to do without help, and to facilitate learning to achieve the goal without support (Brown, Collins, & Duguid, 1989). In McBAGEL, explicit scaffolding of different skills will be provided by expert teachers, through software, and by orchestrating appropriate uses of multimedia tools such as collaboration software, simulation and visualization programs, and decision-support systems. Use of case libraries can also scaffold skills of resource identification and use. Case libraries support student exploration by providing multiple ways of finding and navigating among cases. The skill of searching for relevant information will be scaffolded by presenting the library index in intuitive formats, and encouraging students to explore the library by constructing complex search queries with multiple index terms and conducting searches.

2. Feedback. Providing appropriate feedback is important. One of the key features of problem-solving in the real world is that the problem-solver takes some action and receives feedback from the real world. Research in CBR and cognitive psychology also point out the importance of feedback in coaching (Lesgold, 1994) and in learning from experience. Individuals use feedback to judge how effective their problem-solving efforts were. Feedback can take many forms--for example, it may come from the results of an experiment, the success (or failure) of building and testing working models, or modeling a process by constructing and running a simulation. Previously solved cases may be used to provide anticipatory feedback such as warning the problem solver of a potential problem with a solution that was tried previously. McBAGEL suggests providing students with feedback in a variety of forms and range of fidelity - feedback from teachers, feedback from peer groups, feedback from prior experiences encoded as cases, evaluations by experts on-site or on-line, feedback from computer simulations, and feedback from the real world when working models are built and tested.

3. Emphasis on design problems. Several aspects of design problems make them ideal for promoting learning. (1) They are generally under-specified but might have some over-constrained parts, making understanding and problem definition crucial to

the solution. (2) They have specified clients or audiences who must be satisfied, making the relevance and human dimension of the problem clear. (3) Problems in design are not operational in that a clear-cut path through the problem space is not generally available. Multiple routes to differing destinations exist for the student and the way through is something they must address and navigate. Successful solution requires exploration, questioning, and evaluation. (4) Successful design is iterative. Several to many alternatives are attempted, partial solutions are explored, dry-runs and evaluation occur in the process until several rounds result in a movement forward toward a solution. Students who tend to be task- or product-driven can benefit greatly from participating in such a process. (5) In moving through a design space, numerous criteria have to be managed simultaneously. In any manufacturing or construction problem, for example, cost, manufacturability, availability of materials, time, and environmental issues co-occur as constraints to be tackled. Managing this complexity mirrors what students often know about problems in the real world, lending a sense of authenticity to the tasks while distinguishing design problems from back of the book problem solving. (6) Real-world design problems often have several parts that interact in interesting ways and that require a variety of different kinds of domain knowledge to be solved. When parts of problems interact, there is usually no one right way to address the interaction; rather, successful solution depends on considering the many ways of trading off interactions against each other and choosing between alternatives. (7) The need for interaction with the real world makes design problems nice as well; design problems are situated in the world we live in, making their relevance and the relevance of the math and science needed to solve them clear. At the same time, making a solution operational requires interaction with the real world and thus affords feedback.

4. Collaboration. Collaboration is a key piece of our pedagogy. Research on collaborative learning shows that learning while solving problems in groups facilitates the learning of articulation skills, makes learning more effective for all group members, and allows students to successfully tackle problems more complex than any one group member could individually solve (Brown & Palincsar, 1989; Pea, 1993; Vye et al., 1995). Aspects of our approach - the division of the student body into small groups, the complexity of the design problems that the groups will tackle, and the use of collaboration software to scaffold communication and cooperative work - are all intended to overcome these limitations and enhance the benefits of group-oriented learning. In our approach, scaffolding will be provided to help students learn *to* collaborate as well as learn *through* collaboration.¹

5. Reflective articulation. An important aspect of our approach is promoting reflective articulation - articulation that engenders reflection, leading to enhanced understanding. There are several forms of reflective articulation including generating analogies, predicting outcomes of events or processes, developing questions about the learning materials, and self-explanations (Chi et al., 1989; Weinstein & Mayer, 1985). Reflective articulation can enhance retention, elucidate the coherence of current understanding of the problem being solved, improve self-directed learning skills, and provide a mechanism for abstracting knowledge from the content in which it was learned, thus facilitating transfer. McBAGEL emphasizes two levels of reflective articulation - individual and group - within a collaborative learning environment.

6. A shared electronic workspace. We believe that an electronic workspace which seamlessly integrates a full variety of functionalities, tying together tools that students will use for collaboration, communication and problem solving, will significantly enhance learning. Such an environment is an ideal vehicle for providing adaptive software-realized scaffolding of various skills. The functionalities that this workspace will provide include: an electronic notebook with both private and sharable sections; case libraries and other information resources; tools for simulation, visualization, etc.; a tool for communication, collaboration, and multimedia document sharing; and a set of basic tools such as word processing programs and spreadsheets. Such an integrated computer-based learning environment that students use as a "professional workspace" is a central component of McBAGEL. It is not sufficient to confine such an environment to a laboratory or classroom. Instead, it needs to be made available to students across courses and across campus (e.g., available in all public computer labs) for providing easy access at all times as well as continuity across the curriculum. This availability should encourage both synchronous and asynchronous collaborative work among students.

7. The centrality of teachers. Helping teachers learn to become expert facilitators and partners in the development process is critical (CTGV, 1993). With student-centered learning, the role of the teacher is increasingly important in facilitating student learning and orchestrating classroom activities. The commitment of teachers to student-centered learning is crucial (e.g., Barrows, 1994). Teachers have many roles in our approach - as expert facilitators of problem solving, learning and collaboration; as full partners in the orchestration of classroom activities; as integrators ensuring that the use of computers and software tools is naturally integrated with other activities; and as cognitive diagnosticians.

There are many challenges in achieving the goals we have described. Students do not necessarily view issues from multiple perspectives nor do they collaborate well. They will often recall rather than reflect. An expert facilitator can help promote these processes but often large class size precludes the small group work that would afford these experiences. We believe that providing computational support - making the right kinds of software and on-line information available for use at the right times and seamlessly integrating the use of computers for communication and problem solving - can alleviate some of these difficulties.

5. Computer Support for Learning in McBAGEL

The software environment for McBAGEL (Narayanan, et al., 1995) needs to address several key requirements: access to information resources such as case libraries; support for synchronous and asynchronous collaboration; support for reflective articulation; and provide tools to support problem solving. Figure 1 is a schematic diagram of its architecture. This environment provides an external memory for keeping track of problem specifications, important facts and constraints, ideas about how to deal with the specifications, and learning requirements. The main screen provides several fields for keeping track of multiple sources of information, design alternatives, and further actions to be taken. Space is provided to record the facts and constraints that are important, to record ideas about how to deal with the specifications, and to keep track of what else needs to be learned, what information needs to be collected, and what actions need to be taken. Together, these windows allow the student to see where s/he is now, where s/he has been, and where s/he is going. This screen can be used as an individual workspace or as a shared workspace for the group. The main screen (Figure 2) also provides access to other resources and tools that students need to solve the design problems: case libraries and other information resources; tools for simulation, visualization, decision making etc.; a tool for inter- and intra-group communication, collaboration, and multimedia document sharing; and a set of basic tools such as document processing programs, drawing/painting programs and spreadsheets.

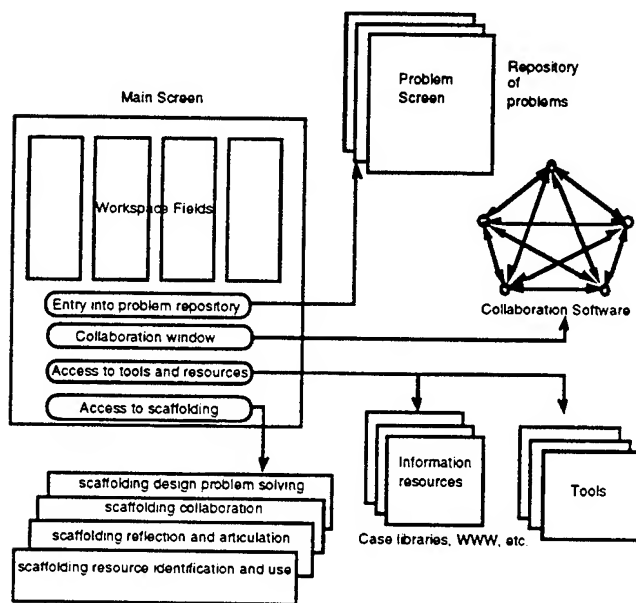


Figure 1. Software Architecture

The problem screen (Figure 3) provides easy access to the evolving problem description. This screen begins with a minimal description of the design problem presented to the students. Details emerge as they inquire about additional information on constraints, material resources and functional issues regarding the design. The collaboration window allows students to enter into a collaboration environment that provides much more than mere communication facilities. It will provide an ability to enter into structured discussions on different topics pertaining to the class and the problem at hand as well as to share multimedia resources with other members of the group and class. A user will be able to browse through past and ongoing discussions which are presented in a structured format to allow easy topic-based, time-based or author-based browsing, and to contribute to those discussions by constructing and sending different types of messages. This collaboration facility will be made available not only to students, but also to teachers. It will provide teachers with a means to collaborate in conducting a course and to share experiences and learn from each other. It can also be a vehicle for student assessment based on their collaborative interactions.

collaboration, and reflection. Design scaffolding will vary as a function of the design stage students are working on. For example when the students are working on problem formulation, the software will provide coaching to help them understand what is involved in this stage: e.g., identifying the problem, formulating the problem, partitioning/decomposing the problem, and framing the problem. The collaboration software will provide procedural facilitation to aid in the development of collaboration skills. Reflection will be facilitated through the articulation that occurs during collaborative problem solving and learning activities.

In summary, this environment will provide ways to organize and manage projects from the students' perspective (e.g., the main screen provides for explicitly listing organizational and learning issues) and the teachers' perspective (e.g., tracking student progress and keeping records of student work). In addition, we envision that the environment will be used for research purposes (e.g., archiving data on inter- and intra-group communications and resource sharing for later assessment, collecting data to be used for student/group modeling in order to devise better course- and student-specific on-line scaffolding and coaching methods, etc.). An initial prototype of this environment has been developed with Hypercard on the Macintosh platform, but it has not yet been tested in a classroom. Borrowing from the metaphor of the white board workspace of problem-based learning found in medical schools, this prototype provides an electronic workspace that is split into four regions. It also allows easy access to other tools and resources. Figures 2 and 3 show the workspace and problem screens of this prototype.

Here is a brief scenario to illustrate how we imagine the students will use this environment. Students, who will be working in small groups, enter the environment at the main screen (Figure 2), which represents their shared electronic workspace. They are provided with relevant information on the design problem they need to solve via the button "problem information". In this case, it is to design an archery stadium for the Olympics. As students are initially formulating and understanding the problem, they will be encouraged to identify data relevant to the problem from the information they have been provided with, and to

articulate this by recording those in the "facts" space. Similarly, as they consider alternative solutions, they will make use of the "ideas" space. The problem-based learning methodology that this environment embodies explicitly prepares students for self-directed learning by requiring them to identify their knowledge deficiencies in the "need to learn" space and the actions they plan to take to remedy those deficiencies in the "action plan" space. Several buttons are found on the bottom of the screen that provide access to different tools that they will need to solve the problem. "Stage" is a pull-down menu which acts as a gateway to various kinds of software-realized scaffolding tailored to different stages of problem solving (Guzdial, 1994).

Figure 3. McBAGEL Problem Screen

While the structure of this environment is still evolving, some of its components have already been designed, implemented and individually fielded in classrooms. In the remaining two sections we elaborate on these implemented components and describe future directions for our research.

6. Implemented Components

• **Case Libraries:** Case libraries organize cases in ways that make it easy to access their most interesting parts, understand their implications, and recognize the range of problems needing solving and the range of solution methods available. Case studies are structured in terms of overviews, problems, stories, and responses. Each story discusses some problem that arose in designing some artifact, the way that the problem was addressed, and the outcomes that resulted. To make it easy for users to extract from stories their important points, stories are presented with illustrative graphics, and several kinds of contextual information is associated with each story. Students can examine the full artifact that some story is associated with, can see a general description of the problem the story addresses, a general description of the kind of solution it provides, and can ask to follow links to other stories that illustrate a similar problem or solution. The stories help students discover which issues they should be considering during design and help them to anticipate the results of carrying out their proposed designs. We have developed a number of case libraries in support of design problem solving.

• **Case Library Authoring Tool:** DesignMUSE (Domeshek & Kolodner, 1992) is a case library authoring tool that has been developed to allow easy construction of case libraries. Student-faculty teams have used it to create case libraries in the domains of architectural design and sustainable technology. While existing case libraries act as information resources, this authoring tool will allow students to construct their own case libraries to record the design problems that they solve.

• **Collaboration support:** The McBAGEL screens themselves provide support for synchronous collaboration by giving the students a shared context for discussion. Our asynchronous collaboration software CaMILE (Guzdial, Rappin, & Carlson, 1995), based in principle on CSILE (Scardamalia et al., 1989), integrates information-gathering tools, communication tools, and applications into a collaborative environment. CaMILE provides a discussion environment into which the full range of text, graphics, spreadsheets, video, and so on that reside locally or on the Internet can be incorporated. It is designed to meet two goals. First, it serves as a collaboration and information indexing tool. Discussions are structured and annotated with links to material anywhere across the network. Second, it serves as a design support tool. Discussions about design problems can be annotated with links to actual ongoing designs. The discussion trace can then serve as a design rationale and a case study of a design. It allows students to collaborate in learning and problem solving by providing a facility for structured inter-group and intra-group communications that are archived, and by providing a way to share multimedia documents easily among collaborators. Like CSILE, CaMILE scaffolds collaboration through procedural facilitation. While electronic mail merely allows team members to share ideas, CaMILE helps them to organize their ideas into coherent arguments, relate their ideas to one another, and use resources across the network to support their arguments. CaMILE can be accessed via the World Wide Web.

• **Exploratory Simulations:** We are developing a range of exploratory simulations (e.g., Rappin, Guzdial, Ludovice, & Realff, 1995) that enable students to learn through simulated experience. Key to these simulations are tight integration with real world problems that the students will be solving, and flexible specification of simulation choices to allow for creative and sophisticated simulation problem solution with immediate feedback. Simulations linked to the middle school science and math problems we are developing are being built with applications like the Logo Microworlds² on Macintosh computers.

7. Promises, Pitfalls and Research Directions

Our approach facilitates the acquisition of factual knowledge in the contexts in which it is likely to be used later in workplaces. This is accomplished by both the use of real-world problems in classroom activities and the provision of multiple real-world cases. Cases, being rich knowledge structures that explicate both conceptual and strategic knowledge, will allow the students to master concepts, principles and strategies in the course of attempting to solve problems. This should promote transfer. As cases will be connected to domain principles, the learner will be able to understand how knowledge is applied to problems, and this should in turn lead to the acquisition of flexible knowledge. Case libraries also facilitate the acquisition of prior examples to apply in later situations. The collaborative nature of student activities should facilitate the assimilation of new knowledge since it encourages articulation and intra-group communication. All these factors should together also result in longer term retention, and successful application of the learned knowledge beyond the classroom.

The major pitfall that has been identified in the use of CBL is the inflexibility of the knowledge and strategies acquired (Williams, et. al., 1993). In the anchored instruction environment, when students were asked to solve a related problem, they often used the original solutions and did not appropriately adapt it to the new situation. When students were asked to predict the solution to "what-if" variants of the problem and received feedback via a simulation of the problem situation, transfer became more flexible as students developed more accurate causal models of the problems (Williams, 1994). In the McBAGEL approach, rather than having students revisit a single case, we have students revisit ideas from multiple cases both through the design problems that students work on and the cases in the case libraries. Students will reflect on, and articulate, how different cases are similar and different to the problems. The problems that students solve are design problems which we believe require a different variety of reasoning strategies from the diagnostic problems traditionally used in PBL. The design cases that are made available to students to use during problem solving contain information both in the form of problem-solution pairings as well as elaborations such as causal models and theoretic justifications. All these factors ought to enable students to construct flexible and coherent knowledge structures and grasp multiple problem-solving strategies. McBAGEL also provides a long-term record of students' problem-solving thus allowing them to revisit their earlier experiences in later efforts.

Issues of knowledge flexibility can be addressed by attending to the kinds of learning strategies encouraged in CBL. Bassok and Holyoak (1993) make a distinction between top-down and bottom-up learning. Top-down learning depends on prior knowledge of the domain coupled with active learning strategies that allow the learner to make principled judgments about the importance of features to the learner's goals. Bottom-up learning refers to inductive learning by examples. Bottom-up learning requires that students make generalizations from multiple examples. The learners do not engage in deep analysis of principles and may end up knowing sets of correlated features (including some that are irrelevant). To the extent that top-down learning enables learners to successfully identify relevant but non obvious features of a problem, more flexible transfer will be promoted (Bassok & Holyoak, 1993; Patel & Kaufman, 1993). Several studies have demonstrated that PBL students are likely to take a hypothesis-driven approach to their own learning and thinking (Hmelo, 1994; Hmelo, Gotterer, & Bransford 1994; Patel, Groen, & Norman, 1993). Research on learning and transfer suggests that a hypothesis-driven or top-down approach to learning may be advantageous (Chi, Bassok, Reimann, Lewis, & Glaser, 1989; Bassok & Holyoak, 1993). If the domain knowledge is fragmented however, students may need to have their attention directed to goal-relevant aspects of the problem.

The McBAGEL approach, as it is being further refined and implemented, provides a research opportunity to test our theories of learning. There are two sets of interrelated issues: those related to knowledge and strategy use and those related to the technology that we will use to scaffold learning and transfer. The issues of flexible knowledge and strategy use are critical for understanding the success of this approach. Understanding the way that learners use cases to help them in problem-solving is an important research issue that will help us develop activities that facilitate using cases as effective learning tools. There are also technology issues related to the types of scaffolding and tools provided for the students. We need to better understand how these tools are best integrated and implemented from a cognitive perspective. Some of the related questions which we plan to address as part of our ongoing research are the following. What are students learning? What are the different ways in which cases can be used most effectively in the classroom, in conjunction with student-centered problem solving activities? How can the influence of cases in promoting learning, integration, retention and transfer be accurately measured? What are the disadvantages of using cases and problems in instruction? What makes for an effective case? How can technology be used for effective and adaptive scaffolding? What kinds of student-teacher, student-student, student-computer and teacher-computer interactions should be supported?

8. Conclusion

Case-based learning environments have the potential for helping students to construct usable knowledge and to learn strategies that prepare them for a lifetime of learning. To afford generative learning, the environments need to contain rich sources of information. In addition, opportunities for student articulation and reflection must be provided to help students think deeply about the problems they are working on and to learn to go beyond the problems given. The McBAGEL approach addresses these issues by situating learning in design problems and by providing an integrated learning environment that contains case libraries, simulations, and other tools for learning. By combining the best aspects of other CBL models with new ideas, and by

drawing from research in Education, Cognitive Psychology and Case-Based Reasoning, McBAGEL represents a multi-disciplinary approach to innovation in educational practice.

Acknowledgments

Research reported here has been supported by ARPA under contract N00014-91-J-4092 monitored by ONR, ONR under contract N00014-92-J-1234, and by the Woodruff Foundation's support of the EduTech Institute.

References

- Adams, L., Kasserman, J., Yearwood, A., Perfetto, G., Bransford, J., & Franks, J. (1988). The effect of fact versus problem oriented acquisition. *Memory & Cognition*, 16, 167-175.
- Barrows, H. S. (1994). *Practice-base learning: Problem-based learning applied to medical education*. Springfield IL: Southern Illinois University.
- Barrows, H. S. (1985). *How to design a problem-based curriculum for the preclinical years*. NY: Springer.
- Bassok, M., & Holyoak, K. J. (1993). Pragmatic knowledge and conceptual structure: Determinants of transfer between quantitative domains. In D. K. Detterman & R. J. Sternberg (Eds.), *Transfer on trial: Intelligence, cognition, and instruction*, (pp. 68-98). Norwood NJ: Ablex.
- Blumenfeld, P. C., Soloway, E., Marx, R. W., Krajcik, J. S., Guzdial, M., & Palincsar, A. (1991). Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist*, 26, 369-398.
- Bransford, J. D. & Johnson, M.K. (1972). Contextual prerequisites for understanding: Some investigations of comprehension and recall. *Journal of Verbal Learning and Verbal Behavior*, 11, 717-726.
- Brown, A. L., & Palincsar, A. S. (1989). Guided, cooperative learning and individual knowledge acquisition. In L. B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, (pp. 393-451). Hillsdale NJ: Erlbaum.
- Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher*, 18(1), 32-41.
- Brooks, L. R., Norman, G. R., & Allen, S. W. (1991). Role of specific similarity in a medical diagnostic task. *Journal of Experimental Psychology: General*, 120, 278-287.
- Catrambone, R. & Holyoak, K.J. (1989). Overcoming contextual limitations on problem-solving transfer. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 15, 1147-1156.
- Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.
- Cognition and Technology Group at Vanderbilt (CTGV). (1993). Anchored instruction and situated cognition revisited. *Educational Technology*, 33(3), 52-70.
- Cognition and Technology Group at Vanderbilt. (1990). Anchored instruction and its relationship to situated cognition. *Educational Researcher*, 19(6), 2-10.
- Domeshek, E. A., & Kolodner, J. L. (1992). A case-based design aid for architecture, In *Proceedings of the Second International Conference on Artificial Intelligence and Design*.
- Dooling, D. J. & Lachman R. (1971). Effects of comprehension on retention of prose. *Journal of Experimental Psychology*, 88, 216-222.
- Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, 15, 1-38.
- Guzdial, M. (1994). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments*, 4, 1-44.
- Guzdial, M., Rappin, N., & Carlson, D. (1995). Collaborative and multimedia interactive learning environment for engineering education. *Proceedings of the ACM Symposium on Applied Computing*, Nashville TN.
- Hmelo, C. E. (1994). *Development of independent thinking and learning skills: A study of medical problem-solving and problem-based learning*. Unpublished doctoral dissertation, Vanderbilt University.
- Hmelo, C. E., Gotterer, G. S., & Bransford, J. D. (1994). *The cognitive effects of problem-based learning: A preliminary study*. Paper presented at the American Educational Research Association annual meeting, New Orleans LA.
- Kolodner, J. L. (1995). From Case-Based Reasoning to Scaffolded Electronic Notebooks: A Journey. *Proceedings Artificial Intelligence in Education*.
- Kolodner, J. L. (1993). *Case-Based reasoning*. San Mateo CA: Morgan Kaufmann.
- Koschman, T. D., Myers, A. C., Feltoich, P. J., & Barrows, H. S. (1994). Using technology to assist in realizing effective learning and instruction: A principled approach to the use of computers in collaborative learning. *Journal of the Learning Sciences*, 3, 225-262.
- Lesgold, A. (1994). Ideas about feedback and their implications for intelligent coached apprenticeship. *Machine-Mediated Learning*, 4(1), 67-80.
- Lin, X. (1994). *Far transfer problem-solving in hypermedia environment: The role of self-regulated learning processes*. Paper presented at the American Educational Research Association Annual Meeting, New Orleans.
- Narayanan, N. H., Hmelo, C. H., Petrushin, V., Newstetter, W. C., Guzdial, M., & Kolodner, J. L. (1995). Computational support for collaborative learning through generative problem solving. *Proceedings of the Conference on Computer Support for Collaborative Learning*, to appear.

- Needham, D. R. & Begg, I.M. (1991). Problem-oriented training promotes spontaneous analogical transfer. Memory-oriented training promotes memory for training. *Memory and Cognition*, 19, 543-557.
- Norman, G. R. & Schmidt, H.G. (1992). The psychological basis of problem-based learning: A review of the evidence. *Academic Medicine*, 67, 557-565.
- Patel, V. L., Groen, G.J., & Norman, G.R. (1993). Reasoning and instruction in medical curricula. *Cognition & Instruction*, 10, 335-378.
- Patel, V. L., & Kaufman, D. R. (1993). *Development of knowledge-based reasoning strategies with medical training* (Technical Report CME93-CS3): Cognitive Studies in Medicine: Centre for Medical Education.
- Pea, R. D. (1993). Practices of distributed intelligence and designs for education. In G. Salomon & D. Perkins (Eds.), *Distributed cognitions*. New York: Cambridge.
- Pirolli, P., & Recker, M. (1994). Learning strategies and transfer in the domain of programming. *Cognition and Instruction*, 12, 235-275.
- Perfetto, G. A., Bransford, J.D., & Franks, J.J. (1983). Constraints on access in a problem solving context. *Memory & Cognition*, 11, 24-31.
- Rappin, N., Guzdial, M., Ludovice, P., & Realf, M. (1995). DEVICE: Dynamic environment for visualizations in chemical engineering. In *Proceedings of the AI-Education Conference*.
- Scardamalia, M., Bereiter, C., McLean, R.S., Swallow, J., & Woodruff, E. (1989). Computer-Supported Intentional Learning Environments. *Journal of Educational Computing Research*, 5, 51-68.
- Schank, R. (1982). *Dynamic memory: A theory of learning in computers and people*. New York: Cambridge.
- Schmidt, H. G., DeGrave, W. S., DeVolder, M. L., Moust, J. H. C., & Patel, V.L (1989). Explanatory models in the processing of science text: The role of prior knowledge activation through small group discussion. *Journal of Educational Psychology*, 81, 610-619.
- Spiro, R. J., Coulson, R. L., Feltovich, P. J., & Anderson, D. K. (1988,). Cognitive flexibility theory: Advanced knowledge acquisition in ill-structured domains. *Proceedings Tenth Annual Conference of the Cognitive Science Society*. Hillsdale NJ: Erlbaum.
- Vye, N. J., Goldman, S. R., Voss, J. F., Hmelo, C., & Williams, S. (1995). Complex math problem- solving by individuals and dyads: When and why are two heads better than one?. submitted for publication.
- Weinstein, C. E. & Mayer, R.E. (1985). The teaching of learning strategies. In M. C. Wittrock (Ed.), *Handbook of research on teaching*, (Third edition ed., pp. 315-327). NY: MacMillan.
- Williams, S. M. (1993). Putting case based learning into context: Examples from legal, business, and medical education. *Journal of the Learning Sciences*, 2, 367-427.
- Williams, S. M. (1994). *Anchored simulations: Merging the strengths of formal and informal reasoning in a computer-based learning environment*. Unpublished doctoral dissertation, Vanderbilt University.
- Williams, S. M., Bransford, J. D., Vye, N. J., Goldman, S. R., & Carlson, K. (1993). *Positive and negative effects of specific knowledge on mathematical problem-solving*. Paper presented at the American Educational Research Association Annual meeting, Atlanta GA.

¹ Thanks to Jennifer Turns for pointing out the importance of this distinction based on her observations of a pilot course.

² Logo Microworlds is a registered trademark of Logo Computer Systems Inc.

Case Libraries in Support of Design Education: The DesignMuse Experiences

N. Hari Narayanan and Janet L. Kolodner
EduTech Institute, College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

A major part of the reasoning designers do involves accessing old cases and using the lessons learned in those situations to address new problems. DesignMuse is a shell for creating on-line case libraries with built-in browsing capabilities that make cases available to design students in a natural way while they are designing. Archie-2 is the most extensive case library built with DesignMuse, and has been used in Georgia Tech's design studios. Archie-2 provides flexible access to multimedia case representations of buildings: libraries, courthouses and skyscrapers. Susie is a student-built library containing cases about sustainable technology and development. These two case libraries illustrate the two ways in which we believe case libraries can enhance learning in the classroom: (1) by using the case library to search for, analyze, compare and contrast cases that are similar to the problems students are solving, which will improve their problem solving skills, and (2) by constructing cases and building a library after doing research in a domain, which will help them learn domain knowledge. This paper describes our classroom experiences with these two DesignMuse-based case libraries and work in progress on enhancing various functionalities of these libraries based on feedback from students and informal classroom observations.

1. Introduction

Design is an activity that requires not only knowledge of facts and fundamentals, but also the spark of creativity and insight. In order to generate innovative solutions to design problems, expert designers often turn to the history of failed as well as successful designs as an invaluable information resource. For novice designers, such information is even more valuable because for them it can serve not only as a source of ideas but also teaches them about the impact that good and bad designs have had in the past, different perspectives that were brought to bear on trend-setting designs, and the plethora of issues that arise during design. Thus, previous designs can be a rich source of ideas and inspiration in the early stage of design when the design problem is still open-ended and evolving. Another fact to note is that design in the real world has increasingly become a multi-disciplinary group-oriented process in which multiple perspectives converge - aesthetic, ergonomic, economic, technical and social, to name a few. Often, successfully addressing issues that arise during design requires an understanding of the interactions between those issues and an ability to arrive at appropriate tradeoffs. All these strongly suggest that collecting, organizing, indexing and presenting design cases structured in a way that highlights the relevant issues involved and lessons that can be learned from them, and making these available as an on-line library of design experiences, can be a very powerful aid for both design students and expert designers. Therefore, design case libraries and case library authoring tools have become an important part of the design education initiative of EduTech Institute, Georgia Tech. The use of case libraries for education derives from research on case-based reasoning [5], the use of past experiences to solve current problems. The central idea in case-based reasoning is that cases facilitate the solution of new problems by suggesting applicable past solutions, pointing the way out of quandaries, allowing potential failures and errors to be anticipated and avoided, and focusing attention on relevant issues.

We believe that design cases can be used in two ways to help students become better designers and learn domain knowledge well: (1) Analyzing past cases while solving design problems can help

In Proceedings of the 25th Annual Conference on Frontiers in Education, Atlanta, GA, 11.95, IEEE Press.

students do design better in a variety of ways: cases illustrate how design problems have been solved in the past, provide warnings about potential pitfalls, focus attention on significant issues, suggest potential solutions, and guide adaptation of earlier solutions to fit the current problem; (2) Constructing new case libraries for others to use helps students acquire deep knowledge of a domain - since cases are actual instances that illustrate the application of knowledge, knowledge thus learned should be transferable to new problems. Archie-2 and Susie are two case libraries which illustrate these two roles. In this paper we describe our classroom experiences with these two libraries and current work on enhancing various functionalities of these libraries based on feedback from students and informal classroom observations.

2. DesignMuse and Case Libraries

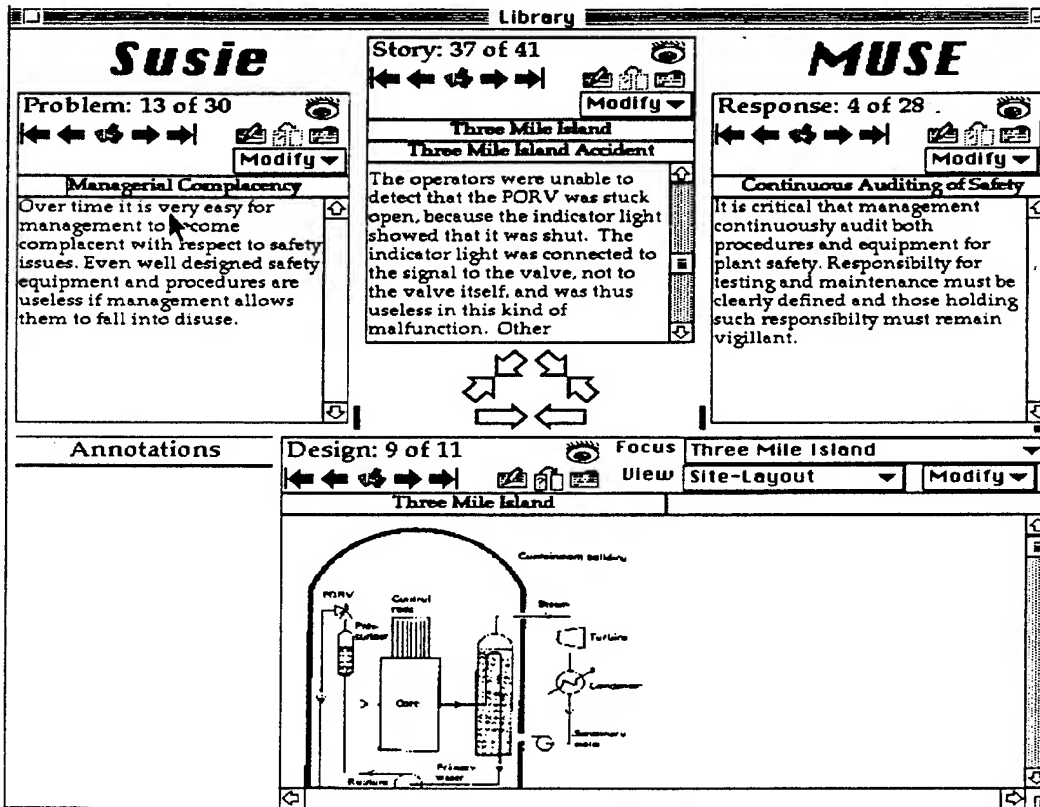


Figure 1. Case Presentation Window

DesignMuse [3] is a case library authoring tool that allows easy construction of structured, indexed and searchable databases of analyzed case studies for students to learn from. DesignMuse builds functionalities for indexing, organizing and presenting case information into the case libraries created from it. It also provides graphical user interfaces for specifying search probes, for presenting case information and for modifying/extending the case library. Cases are structured in terms of stories, problems, responses and design overviews. These four pieces of information are presented in four panes of a case presentation window as shown in Figure 1. Stories help students discover which issues they should be considering and help them to anticipate the results of carrying out proposed solutions. Stories are associated with general problems and responses they illustrate. The presentation format - a story flanked on the left by the problem and on the right by the response - is intended to help students take the abstraction step from specific stories to the general problem-solution pairs that the stories illustrate (and vice versa). Design overviews, appearing in the lowermost pane of the case presentation window, provide appropriate context for stories, problems

and responses. Hypertext links between design overviews and stories, and among stories, problems and responses allow flexible navigation and browsing.

DesignMuse has been used both by experts (faculty, graduate students and working engineers) to construct case libraries for use in design studios and by novice students to construct prototype case libraries. Archie-2, described in more detail later, is an example of the former. DesignMuse was also used by students in a graduate level case-based reasoning course in two quarters. In the first quarter (Winter 1994), groups of students produced six different sample case libraries: (1) a building design case library, (2) a re-design advisor that provides advice about replacing metal aircraft parts with composites, (3) a case guide to formulating design problems in the decision support problem framework, (4) an on-line failure catalogue for diagnosing satellite failures, (5) an advisor for car purchase decisions, and (6) a satellite command management system. During the second quarter of use (Winter 1995) DesignMuse was used by three groups of students to create three parts of a single case library on the topic of sustainable technology and development. One group focused on cases of industrial pollution, another on issues of sustainable development and resource management, and the third on industrial accidents. Susie (SUStainable technology Interactive Education) is the case library that resulted from merging these three parts.

Archie-2 [2] is a building design case library. It provides flexible access to multimedia representations of building designs. Each building has been evaluated in a post-occupancy evaluation, and is represented as a series of plans/montages, design issues/problems that came up during its design, construction and use, appropriate responses to those issues/problems, and stories that illustrate how various design issues/problems were addressed. Each such "chunk" of information is indexed by the physical or functional part of the building it is associated with and the design issues it addresses. This case library contains cases about the design of several existing libraries and courthouses in the Atlanta area. Its database currently consists of 6 library designs, 3 courthouse designs, 67 problems, 187 stories, and 138 responses. Each story in this library discusses some problem that arose in a building design, the way that the problem was addressed, and the outcomes that resulted. Archie-2 was used once in a graduate-level design studio in the College of Architecture. A graduate level class on human-computer interface design at the College of Computing conducted a formal evaluation of the usability of Archie-2's interface. We are now in the process of extending the library by adding cases of tall building designs and building modifications for handicapped accessibility. This extended library will be tested in a graduate-level architectural design studio in the 1995-96 academic year.

Susie is a library of cases about pollution, the natural environment and industrial accidents that highlight issues of sustainable technology and development. It illustrates a different use of case libraries in classrooms: learning, not by perusing an existing library, but instead by creating one. Graduate students in a course on case-based reasoning in the College of Computing built Susie in the Winter Quarter of 1995. It holds 11 large cases, with 30 problems, 41 stories and 28 responses. Its individual stories teach lessons about making technology and development decisions that take into account environmental and ethical concerns as well as the more traditional technological and economic issues. Figure 1 shows the case presentation window of Susie with a story and accompanying problem and response statements about the Three Mile Island accident.

3. Lessons from Classroom Experiences

3.1. Broadening the Content and Presentation of Information

How can cases be organized and presented in ways that make it easy to learn from them? DesignMuse-based case libraries provide one answer to this question in terms of individual information chunks - design overviews, problems, stories and responses. Another, for example, is the "multimedia book" style of engineering design cases developed at the University of California, Berkeley [4]. The majority of information contained in our case libraries are specific instances -

e.g., specific designs and stories. The only relatively abstract information provided are problem and response statements which generalize the lessons to be drawn from specific stories. During the course of extending Archie-2 for use in a tall building design studio, the faculty member in charge of this studio felt that having the case library also present introductory tutorial material as well as specific cases of existing tall buildings would make it more useful to students. This studio consists of two phases: introducing students to the methods, technologies, systems and issues of tall building design, followed by a design project. If tutorial materials were to be incorporated into Archie-2, it was felt that students could use the case library in both phases instead of only during the design project. Work is therefore currently underway to collect and structure introductory materials in the form of problems, stories, responses and designs connected to a "generic tall building case" in the library. In this case, explanatory materials that further illustrate a problem-response pair will replace actual stories, and generic drawings and design guidelines for a building will replace the specific floor plans or montages that typically appear in the design pane (the lowest pane in Figure 1) of the case presentation window. Another extension to Archie-2 in progress is the addition of designs for handicapped accessibility as mandated by the Americans with Disabilities Act (ADA). The case information being added is about redesign of existing buildings to conform to ADA regulations, not new building designs. This has prompted us to take a broader view of the design pane of the case presentation window and to use it to present information such as redesigns of building parts like entry ports and guidelines or regulations that governed the redesigns.

While students were constructing Susie, the one aspect they had the most difficulty with was the extraction of relevant information from the resource materials on sustainable development and technology they were provided with (e.g., newspaper articles, research papers, project reports, reports issued by agencies such as the EPA and Greenpeace) and structuring that information in the form of "chunks" that DesignMuse-based case libraries require - problems, stories, responses and designs. While problems, responses and illustrative stories lend themselves naturally to describing almost any situation, design information is not equally applicable or relevant. For example, one sustainable development case dealt with how to manage forested patches of land called "windbreaks" to prevent soil erosion as well as sustain game for sports hunters. This case has no design component to it; its focus is environmental management. Therefore it was decided to change the design pane to an overview pane that may contain any kind of contextual information - this could be an overview, a layout diagram, a time-line or similar graphical/textual device that provides sufficient context for the problems, stories and responses associated with a case. Thus, Susie displays pie-charts, tables and time-lines in this pane instead of the floor plans and montages of Archie-2.

3.2. Indexing: Vocabulary, Organization and Presentation

Case libraries use a structured indexing vocabulary for tagging information, usually created based on an expert's understanding of the domain. Users (e.g., students) use the indexes for two purposes - to construct a probe or query during database search and to construct a set of descriptive indexes for new cases during library construction. We have found that students are not always familiar with the necessarily technical terms appearing in the indexing vocabulary. This impedes their ability to conduct useful searches and to index new information appropriately. This is motivating the addition of a glossary to the library.

Students' efforts in constructing Susie highlighted a need to broaden the indexing system. DesignMuse was originally built to author "design" case libraries. Therefore it was felt that the indexing system of these case libraries should reflect the salient dimensions of artifact design - *where* (physical and functional components), *what* (design issues), *when* (lifecycle of the artifact) and *who* (stakeholders). Accordingly, the indexing system classifies all index terms into these classes and organizes them in a hierarchical structure for efficient searching. However, cases of sustainable development and technology required a different cut on indexes. These cases needed to be characterized in terms of the type of a case, its spatio-temporal scope, the nature of problems

encountered, the nature of possible solutions, affected parties, lifecycle stages (for technology related cases) and relevant issues. What we have discovered is that as case libraries expand into domains that go beyond design, the organizing structure and vocabulary of the indexing system can change drastically. So flexibility to change the structure of the indexing system as well as the vocabulary ought to be built into the case libraries. On the other hand, giving complete freedom to determine this structure to case library designers may result in the indexes reflecting the personal and professional idiosyncrasies of the designers which need not be apparent to the users. This will diminish the utility of the indexes. This is a tradeoff, the implications of which are yet to be investigated in classroom settings.

At present, the system presents its indexing vocabulary to users as a series of nested pop-up menus that faithfully reflect its underlying hierarchical structure. We have observed that the students get confused by the organization of the hierarchy because they are not familiar with the search and matching algorithms that exploit this hierarchical structure. In fact, students do not need to see this structure at all. Besides, the multiple levels of nested menus that this hierarchical structure engenders result in a cluttered set of pop-up menus that students find difficult to maneuver in. The solution we are implementing is to hide the hierarchy (but leave an option to see it if someone wants to) and instead to present the indexes in a more familiar form such as the indexes in a book.

3.3. Software Integration

Educational tools like case libraries can be integrated into classroom practice by either making them available for optional use by students or making their use mandatory in assignments. Our experience has been that either of these strategies does not fully engage the students. In the former, many students do not attempt to use the system. In the latter, many students use the system to the minimum extent necessary to get the assignments done. In either case, students do not exploit the full potential of the case libraries to facilitate their learning. We believe that in order to motivate and engage the students so that they derive the maximum benefit from case libraries, it is necessary to integrate their use with the rest of classroom activities. We are currently pursuing this idea in two ways. One is to integrate case libraries with software that supports design problem solving in other ways. In one project we are building an interactive problem solving system that the students will use to solve design problems, and integrating a case library with it so that one kind of feedback the problem solver will give students is to bring up the case library with a set of cases relevant to the issue they are pursuing at the time feedback was requested. This way students get started off with a set of interesting cases relevant to their problem, which should motivate them to search and explore further in the library. Another project in the College of Architecture seeks to integrate a sketch-pad with case libraries so that an architect could make sketches to guide the search for relevant cases and annotate the retrieved information. A broader approach to integration is to embed case libraries in a larger computer-based learning and work environment with many other components. Integrating a case library into an electronic workspace that provides tools for communication, collaborative work, simulation, etc. as well, for example, will facilitate the use of cases for multiple purposes. Once relevant cases have been found by searching the case library, that information can be immediately be imported into the workspace and used to solve some aspect of the design problem at hand. Cases can also be imported into collaborative discourse or argumentation. For example, cases provide one type of information a student might point to as justification for some argument, as a potential alternative to a design decision, or as a rebuttal to someone else's design decision.

3.4. Curriculum Integration

There is ongoing work in terms of further curricular use of case libraries. Archie-2, the architectural case library, is currently being extended to include cases about handicapped accessibility and tall building designs. This extended library will be fielded in a graduate-level design studio in the College of Architecture in Fall 1995. Work on the creation of a new case library consisting of product design cases has just begun. This library will be used in early 1996 in a collaborative

product development course being offered jointly by the Industrial Design Program and School of Management at Georgia Tech. Another case library documenting Olympics construction projects in the Tech campus, for use in a design course to be offered jointly by Architecture and Civil Engineering in 1996, is also underway. Finally, in the Spring of 1995, five groups of students in a sustainable technology course constructed five case studies on paper: disinfection of waste water, pulp fiction, the role of PVC, pharmaceuticals with chlorine, and sustainability of chlorine use with refrigeration. We plan to use these cases as seed materials for students to extend Susie during the next offering of this course.

4. Case Libraries in Design Education

How can students learn from case libraries? Cases, being rich knowledge structures that explicate both conceptual and strategic knowledge, will allow students to master concepts, principles and strategies in the course of attempting to solve problems. This will promote transfer. As cases will be connected to domain principles, learners understand how knowledge is applied to problems, and this should in turn lead to the acquisition of flexible knowledge. Students will analyze multiple cases from the case library and reflect on how these cases are similar and different to the problems they are solving. Case libraries also facilitate the acquisition of prior examples to apply in later situations. There are two different and equally important ways in which students will use case libraries. One is by searching for, analyzing, comparing and contrasting cases that are similar to the problems they need to solve. The other is by constructing cases from domain knowledge and from their problem solving experiences and incorporating these into the case libraries.

Use of case libraries can also scaffold skills of resource identification and use. The goal of scaffolding is to help students to carry out a reasoning process or achieve a goal that they would not be able to do without help, and to facilitate learning to achieve the goal without support [1]. Case libraries support student exploration by providing multiple ways of finding and navigating among cases. Students might begin looking at one story of interest and explore related stories by a number of different dimensions (e.g., same subsystem being designed, similar challenges, similar solutions), or begin by browsing all the stories about the same subsystem, or one of a number of other dimensions. The skill of searching for relevant information can be further scaffolded by presenting the library index in intuitive formats and encouraging students to construct and experiment with complex search probes from the index terms.

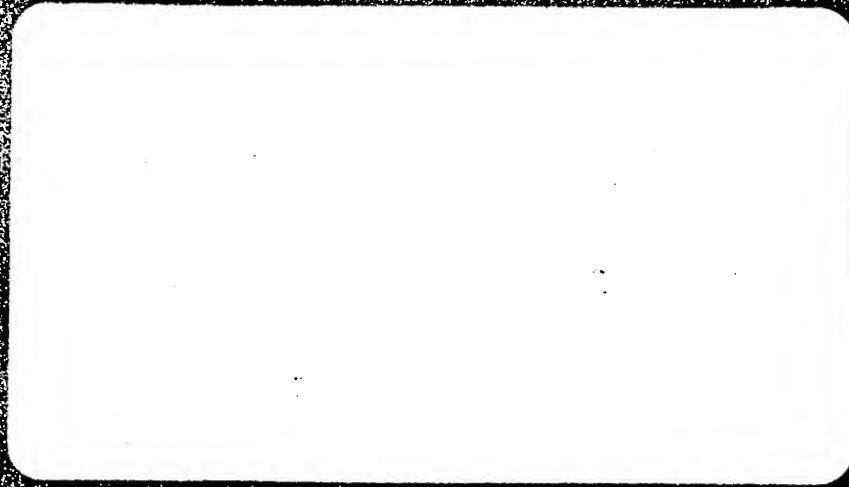
Acknowledgments

Research reported here has been supported by ARPA under contract N00014-91-J-4092 monitored by ONR, ONR under contract N00014-92-J-1234, and by the Woodruff Foundation's support of the EduTech Institute. Research on Archie-2 has been conducted in collaboration with Craig Zimring, College of Architecture, Georgia Tech.

References

1. Collins A., Brown J.S., & Newman S.E. (1989). Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. In: Resnick, L. B., ed. *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*. Hillsdale, NJ: Lawrence Erlbaum and Associates.
2. Domeshek, E.A., & Kolodner, J.L. (1992). A case-based design aid for architecture. In Gero J, ed. *Proceedings of the Second International Conference on Artificial Intelligence and Design*.
3. Domeshek, E. A., Kolodner, J. L., & Zimring, C. M. (1994). The design of a tool kit for case based design aids. In Gero, J. (ed.) *AI and Design 94*. (in press).
4. Hsi, S., & Agogino, A. M. (1994). The impact and instructional benefit of using multimedia case studies to teach engineering design. *Journal of Educational Multimedia and Hypermedia*, 3(3/4), 351-376.
5. Kolodner, J. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

COGNITIVE SCIENCE REPORT SERIES



Cognitive
Science

GIT-COGSCI-95/02

**Beyond Domain Knowledge:
Towards a Computing Environment for the
Learning of Design Strategies and Skills**

Ashok K. Goel
Andres Gomez de Silva Garza
Nathalie Grue
Margaret M. Recker
T. Govindaraj

November, 1995

Ashok K. Goel
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332
goel@cc.gatech.edu

Beyond Domain Knowledge: Towards a Computing Environment for the Learning of Design Strategies and Skills

Abstract

Although interactive computing systems potentially offer a flexible environment for teaching and learning, they have had surprisingly little impact on the content or methods for teaching design skills. Most commercially available computer-based design tools support only some "low-level" tasks such as drafting and drawing, numerical computing, and constraint propagation. These tools do not support the "high-level" cognitive task of conceptual (or preliminary) design. In our own earlier work, we have developed a sequence of prototypical systems for supporting conceptual design. While these experimental systems apparently are very useful for supporting design problem solving and are intended to be used by practitioners, their utility for the teaching and learning of design seems quite limited. These systems essentially provide an external memory of domain knowledge (*e.g.*, primitive objects and relations in the domain), and design experiences (*e.g.*, design cases that specify the problem, the solution, and the outcome of a past design experience). This external memory of past design cases evidently helps designers in solving new design problems in a given domain. Further, in contrast to traditional methods of teaching, these systems offer a designer opportunities to learn domain knowledge in the context of concrete design examples and problems. However, these systems do not provide potential users with access to knowledge of design tasks and problem-solving methods. They assume that the user is an expert designer who already knows what tasks need to be set up in solving a design problem, and what problem-solving methods are useful for addressing a given task. We believe that a computing environment for teaching design to novice designers must enable the learning of design tasks and problem-solving methods in addition to the learning of domain knowledge and design cases. Our group is developing an interactive computing environment called CANAH-CHAB to address some of these issues. The goal of this paper is to analyze some of the issues in building a computer-based system for teaching and learning of design strategies and skills, and to describe how these issues have influenced the design of CANAH-CHAB.

1 Goal and Motivations

Traditional methods for the teaching of design skills place primary emphasis on learning: (1) domain knowledge, and (2) analytical approaches for using this knowledge in refining design solutions (e.g., [Vincenti 1990]). In addition, domain knowledge, e.g., physical principles and engineering mechanisms, is typically taught in the abstract, sometimes augmented with textbook examples. Then, during exercise and evaluation phases, the student is given a design problem in which the analytical skills that have been learned are applied and tested. The problem is usually highly structured and there is little freedom in developing new designs. For example, in a typical problem in mechanical design, a general design solution is given and the student is asked to compute the parameter values that will result in the designed system meeting some given specifications (e.g., [Shigley 1977]). It might be argued that such teaching methods were the inevitable result of the limitations of the "paper and pencil" approach to the teaching of design skills.

In recent years, computers and computer-based tools have entered the modern classroom. Although interactive computing systems potentially offer a much more flexible environment for teaching and learning, they have had surprisingly little impact on the content or methods for teaching design skills. Current computer-aided methods for teaching still emphasize learning domain knowledge and analytical approaches, and domain knowledge still is taught in the abstract. Most commercially available computer-based design tools support only some "low-level" tasks such as drafting and drawing, numerical computing, and constraint propagation. While these provide a limited degree of flexibility in sketching, analyzing, and refining design solutions, these tools do not support the "high-level" cognitive task of conceptual (or preliminary) design. This phase, we claim, results in the generation of design concepts and involves much of the innovation and creativity in design.

More recently, some design researchers have developed experimental computing environments for supporting high-level conceptual design. For example, JANUS [Fischer, McCall, and Morch 1989] is an interactive hypermedia computing environment that provides a designer access to a catalog of past designs and uses this knowledge to suggest arguments for and against specific design choices in a given design situation. Similarly, BOGART [Mostow 1989] is an interactive system that provides a designer access to a catalog of past designs and enables the designer to replay the derivational traces of the past designs. In our own earlier work, we have developed a sequence of prototypical systems for supporting conceptual design: ARCHIE [Pearce *et al.* 1992], a case-based system for supporting architectural design; ASKJEF [Barber *et al.* 1992], an interactive multimedia case-based system for supporting interface design; and ARCHIE TUTOR [Goel *et al.* 1993], an interactive multimedia case-based environment for supporting the teaching of architectural design.

While these experimental systems apparently are very useful for supporting design *problem solving* and are intended to be used by practitioners, their utility for the teaching and learning of design seems quite limited. These systems essentially provide an external *memory* of domain knowledge (e.g., primitive objects and relations in the domain), and design experiences (e.g., design cases that specify the problem, the solution, and the outcome of a past design experience). Some of these also provide a kind of *rationale* for the past designs.

For example, JANUS provides arguments for specific design choices in the past designs, BOG-ART provides the derivational traces of the past designs, and ASKJEF provides a trace of the evolution of the design through cycles of design criticism and redesign. This external memory of past design cases and associated design rationales evidently helps designers in solving new design problems in a given domain. Further, in contrast to traditional methods of teaching, these systems offer a designer opportunities to learn domain knowledge in the context of concrete design examples and problems.

However, these systems do not provide potential users with access to knowledge of design tasks and problem-solving methods. They assume that the user is an expert designer who already knows what tasks need to be set up in solving a design problem, *e.g.*, design verification, and what problem-solving methods are useful for addressing a given task, *e.g.*, qualitative simulation for the task of design verification. This assumption does not seem valid for all users. In particular, it appears invalid in the case of novice designers such as beginning design students. We believe that a computing environment for teaching design to novice designers must enable the learning of design tasks and problem-solving methods in addition to the learning of domain knowledge and design cases.

Further, although these systems provide access to some kind of rationale for the past designs, they do not provide potential users with high-level languages for representing the reasoning activity involved in the creation of the current design. They view design primarily as a problem-solving activity and assume that the design process is over once a satisfactory design has been created. However, design is also a learning activity; both expert and novice designers learn from their design experiences. Furthermore, *reflection* over the design activity, both during and after problem solving, often plays an important role in learning from design experiences. We believe that a computing environment for teaching design must provide high-level languages for representing the design activity so as to enable reflection both during and after design problem solving. See also [Collins and Brown 1988].

Our group is developing an interactive computing environment called CANAH-CHAB¹ to address some of these issues. In particular, our goal is to develop an interactive multimedia computing environment for teaching design that (1) enables the user to explore alternative design concepts, (2) enables the user to create and critique her own conceptual designs, (3) enables the learning of domain knowledge and design cases, (4) enables the learning of design tasks and problem-solving methods, and (5) enables reflection both during and after the design process. Our earlier work on computer-based design support systems such as ASKJEF suggests that, in order to be effective, such an environment must itself be capable of autonomous generation and evaluation of design solutions so that, when needed, it can help a student create and critique her own designs. Hence we are building CANAH-CHAB on "top" of an existing autonomous design system called KRITIK [Goel 1991, 1992] that operates in the domain of engineering devices.

The goal of this paper is to analyze some of the issues in building a computer-based system for teaching and learning of design strategies and skills, and to describe how these issues have influenced the design of CANAH-CHAB, which is presently under development. The next section briefly describes the KRITIK and ASKJEF systems from which CANAH-

¹"Canah" and "chab" are Mayan words that roughly translate to "to learn" and "to design," respectively.

CHAB evolves. Section 3 discusses some of the issues that arise in computer-based teaching and learning of design strategies and skills, and Section 4 describes our hypotheses in regard to these issues. Section 5 outlines the current status of the development of CANAH-CHAB. We conclude with a discussion that relates our work with earlier research and draws some tentative conclusions.

2 Background

As we mentioned above, our work on CANAH-CHAB evolves from our earlier research on both interactive problem-solving and tutoring systems ([Barber *et al.* 1992], [Goel *et al.* 1993], [Pearce *et al.* 1992], [Recker and Pirolli 1993], [Vasandani and Govindaraj 1994]) and autonomous design problem-solving and learning systems ([Goel 1991, 1992], [Goel and Chandrasekaran 1989, 1992], [Bhatta and Goel 1993, 1994]). In particular, it directly builds on the ASKJEF system, which interactively supports software engineers in designing simple human-machine interfaces, and the KRITIK system, which autonomously designs simple engineering devices. Since we will often allude to ASKJEF and KRITIK in this paper, we briefly describe the two systems in this section. We will return to our work on CANAH-CHAB in the next section.

2.1 KRITIK

KRITIK is an autonomous design problem-solving and learning system that operates in the domain of engineering devices such as electrical circuits and heat exchangers. It takes as input the user's specification of the function that is desired, and provides as output a description of the structure of a device that produces the function. It uses multiple types of knowledge in solving design problems, for example, past design cases and structure-behavior-function (SBF) models of how the specific devices stored in the design cases actually work. A design case is indexed by the functions delivered by stored design and acts as an index into the SBF model for the stored design. Given a design problem, KRITIK solves it by first elaborating on the functional specification of the problem, and retrieving from its case memory a design that delivers the function similar to the desired one. It then identifies candidate modifications to the structure of the retrieved design by analyzing SBF model for the design, executes the required modifications on design structure to produce a candidate design, and revises the SBF model of the retrieved design to produce a SBF model for the candidate design. The system then evaluates the candidate design by qualitatively simulating the SBF model. If the design fails then KRITIK attempts to redesign it. If it succeeds, then KRITIK uses the SBF model of the new design as a causal explanation to learn the indices for storing the new design case and associated SBF model in memory, and stores them for potential reuse in the future.

The design of KRITIK is based on a task structure analysis on design problem solving with Chandrasekaran's *task structures* [1990] as the framework for analyzing problem solving. A problem-solving task in this framework is specified by the information it takes as input and the information it produces as output. A task can be accomplished by one or more

methods, each of which decomposes it into a set of simpler subtasks. A method is specified by the subtasks it sets up, the control it exercises over their processing, and the knowledge it uses. The subtasks into which a method decomposes a task can, in turn, be accomplished by other methods. or, if the appropriate knowledge is available, they can be solved directly. This enables KRITIK to opportunistically select a problem-solving method depending on the current subtask it is addressing. Since the system may select different methods for different subtasks, this enables KRITIK to integrate several reasoning strategies such as case-based and model-based reasoning and to shift from one strategy to another depending on the needs of the current subtask. KRITIK's task structure for design is illustrated in Figure 1.

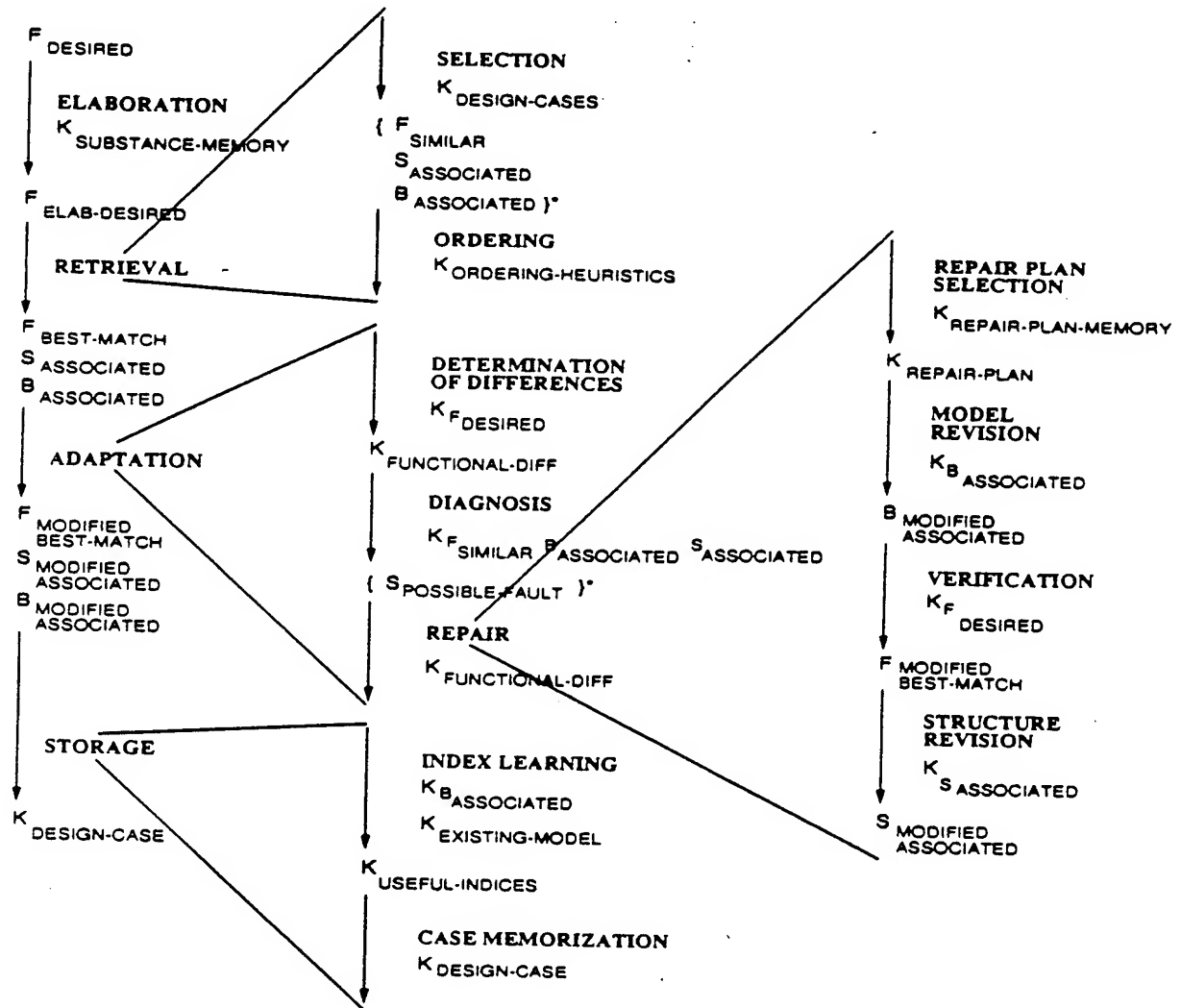


Figure 1: KRITIK's Task Structure.

KRITIK's SBF models of devices capture a different component of the design rationale from the one provided by ASKJEF [Chandrasekaran, Goel, and Iwasaki 1993]. An SBF device model specifies the internal causal behaviors that explain how the structure of a device delivers its function. Figure 2 shows the SBF model of an electric circuit in KRITIK's memory. Lee and Lai [1991] have noted that, in addition to the derivational component

of the design rationale that provides a record of the design activity, the design rationale may also contain an explanation of how the designed artifact works. In other words, one component of design rationale explains how the choices for the design components in the artifact and the relations between them together achieve the desired functionalities of the artifact. KRITIK's case-specific SBF models express precisely this knowledge. We call this the *behavior-centered* component of the design rationale.

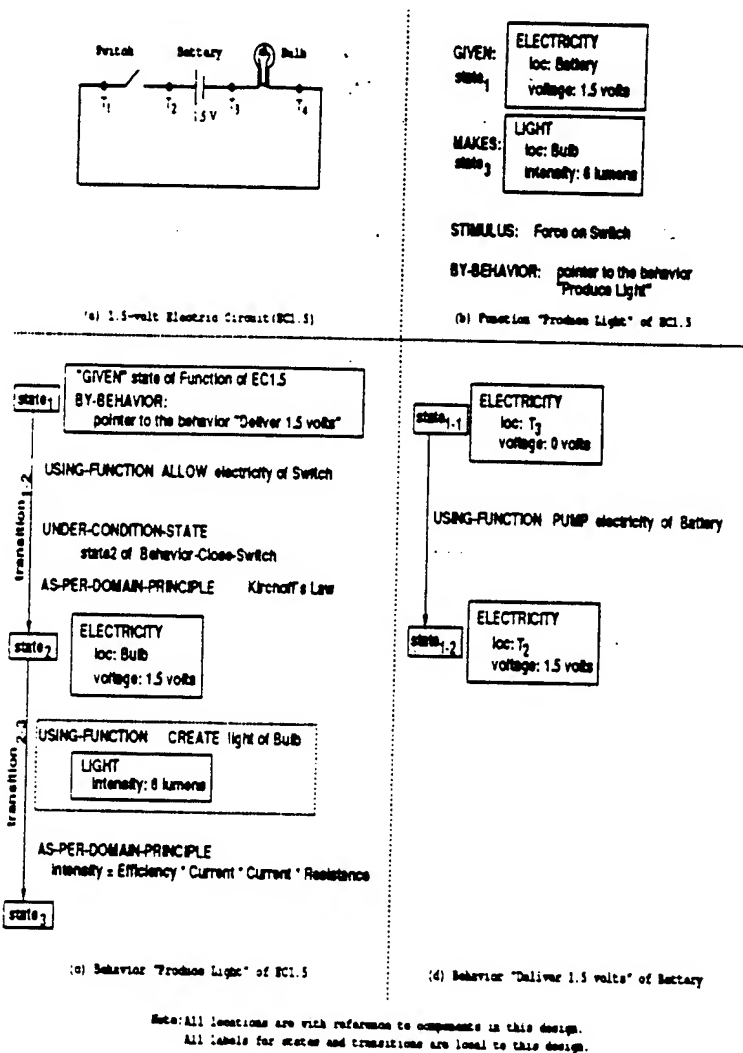


Figure 2: SBF Model of an Electrical Circuit in KRITIK's Memory.

2.2 ASKJEF

ASKJEF [Barber *et al.* 1992] is an interactive multimedia knowledge-based system for advising software engineers on the design of human-machine interfaces. It uses text and graphics for acquiring information about design problems from the software engineer, and it uses text, graphics, animation, and voice for presenting several different kinds of knowledge to

the engineer. In particular, the system provides a software engineer with an external memory of design cases, guidelines, explanations, and stories. These different types of knowledge are cross-indexed in ASKJEF's memory as indicated in Figure 3. This cross-indexing enables the user to navigate through ASKJEF's memory. The system itself uses these different types of knowledge for (1) helping a software engineer understand a given problem by illustrating and explaining previous design solutions to similar problems, and (2) helping the engineer comprehend the principles of interface design by illustrating and explaining the use of interface-design guidelines in the context of design problems encountered in the past.

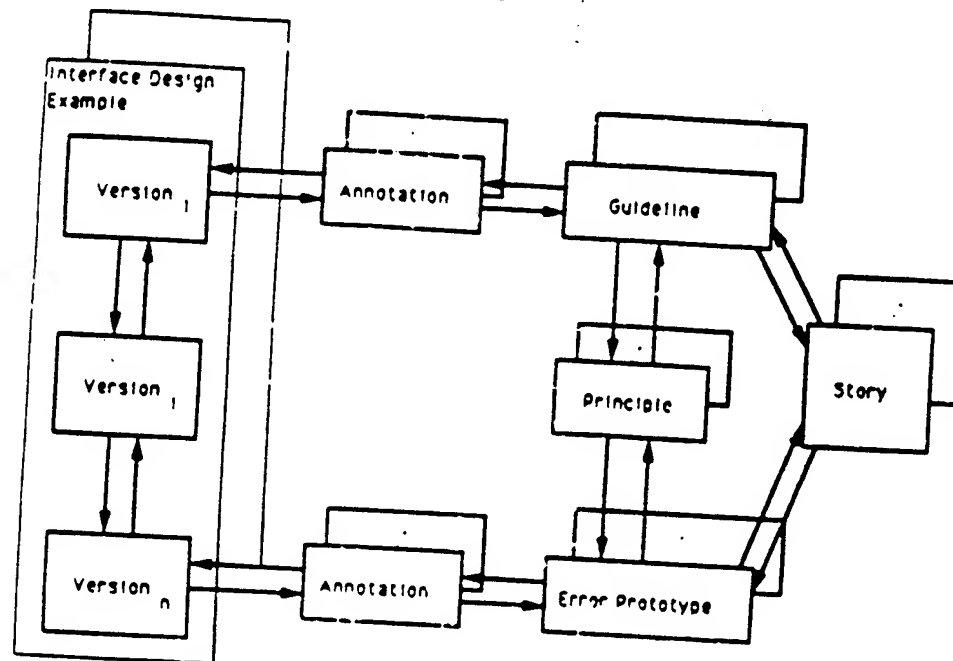


Figure 3: ASKJEF's Knowledge Organization.

As indicated in Figure 3, each interface design case contained in ASKJEF is decomposed temporally and actually contains several versions of the interface design. Every version provides a "snapshot" in the temporal evolution of the interface design. The different versions are annotated with design critiques that specify what worked in a given version, what did not work, and how these "failures" were repaired in the next version. These annotations link the different interface versions together into a partial record of the reasoning that led to the creation of the final interface design. In this way, ASKJEF provides the rationale behind the designs stored in its memory. We call this an *artifact-centered* view of design rationale.

This design rationale in ASKJEF supports interface designers in two ways. Firstly, it supports them in critiquing their designs. Since the design rationale stored with each interface design case explicitly shows the utility of design critiquing in past design situations, it directly points to the need for critiquing in the current design situation. In addition, since the design rationale illustrates the kinds of errors that occurred in a past interface design and the kinds of remedies used to fix them, it helps to guide the critiquing of the current

design. Secondly, design rationale in ASKJEF supports design collaboration over time. An interface design case specifies a design created by interface designers in the past and offers it for reuse in the present. It thus enables collaboration between the past and present interface designers. By making some of the reasoning behind the creation of a past interface design example more explicit, the design rationale for the past design enables the past designers to more effectively collaborate with the present designers in solving new interface design problems.

3 Content of Learning

With KRITIK and ASKJEF as the background, we now turn to the main issues in designing CANAH-CHAB. One of the primary issues in the design of CANAH-CHAB is "what is the content of learning in design, i.e., what learning tasks should the system support?" Earlier systems such as ASKJEF and ARCHIE TUTOR implicitly support the learning of domain knowledge. ASKJEF, for example, provides access to domain knowledge in the form of design examples, design guidelines, and design rationales. This type of domain knowledge seems necessary in order to achieve any degree of design expertise in a given domain. But what other kinds of design knowledge should the interactive system enable a user to learn?

Another primary issue in the design of an interactive learning environment such as CANAH-CHAB is "what are the methods of learning in design, i.e., what methods should the system use to enable the learning of design?" Earlier systems such as ASKJEF and ARCHIE TUTOR take a two-pronged approach to this issue. First, they help a user learn domain knowledge in the process of solving design problems. They do not merely present domain knowledge for memorization by the user. Instead, the designer gets exposed to domain knowledge in the context of solving a real problem. Second, they make extensive use of examples, cases and stories to illustrate and explain abstract domain concepts such as design principles and guidelines. These methods represent a departure from the traditional methods of learning domain knowledge. But what other methods of learning should the interactive system support?

Of course, the design of an interactive system such as CANAH-CHAB also involves complex issues of knowledge representation and organization, human-machine interaction and interface organization, and the use of multimedia and hypermedia. However, we believe that these issues are auxiliary in that they cannot be adequately answered until the issues of the content and methods of learning have been resolved to some degree.

The answer to the issues of content and methods of learning depend on the intended users of the system. Evidently both novice and expert designers learn as they create new designs, and they learn both from their failed and successful design experiences. However, in general the content and methods of learning may differ depending on where a given user lies on the novice-expert continuum. Earlier systems such as ASKJEF and ARCHIE TUTOR assume that the user is an expert design practitioner. In contrast, CANAH-CHAB is intended for novice designers such as beginning design students in engineering colleges and its goal is to help the student make the transition from a novice designer to a design expert. Note that since

4.1 Learning of Design Strategies

The design of CANAH-CHAB incorporates more than one method for supporting the learning of functional knowledge of design tasks and strategic knowledge of problem-solving method. The first learning method is *learning by observing* [Winston 1992]. In "learning by observing," the system illustrates and explains a design task or a problem-solving method by use of an example. Design tasks and problem-solving methods are abstract concepts that are sometimes hard for design students to understand. Thus, it is important for a computing environment to be able to illustrate and explain a given task or method. We propose to do this by using illustrative examples. The user can learn design tasks and methods from such examples because she can observe them in actual use on specific problems, which helps her to ground the acquired knowledge in concrete examples.

This brings us to the benefits of modeling KRITIK's design problem solving in CANAH-CHAB. Since KRITIK is an autonomous design problem solver, it enables us store traces of its problem solving on different design problems in different domains. These traces can be indexed by the types of tasks and methods they use and stored in CANAH-CHAB. When a user asks the system to illustrate a given design task or method, CANAH-CHAB can refer to its memory of traces of problem-solving episodes in order to provide the user with an appropriate example. Note that CANAH-CHAB may not always be able to find any relevant problem-solving episode with which to illustrate a given design task or method. In this case, it can give KRITIK a problem to solve at run time that will potentially illustrate the task or strategy, record the trace of how KRITIK solves the problem, and then present this trace to the user. This represents an added benefit that is obtained by using an autonomous system as a model of a design problem solver. In this way, KRITIK problem-solving capability can be used to complement CANAH-CHAB's knowledge.

Another method for supporting the learning of functional and strategic knowledge is *learning by using*. In "learning by using," the computing environment enables the user to actually use a given problem-solving method in the context of a real problem. This learning method however raises two issues: (i) how to explicitly represent the knowledge of design tasks and problem-solving methods, and (ii) how to organize this knowledge so that it can be accessed when needed. Clearly, without an explicit declarative representation, a system cannot communicate a design task or a problem-solving method to a user. In a different project called AUTOGNOSTIC, Stroulia and Goel [1993, 1994] have viewed problem solvers as abstract devices and how their functioning can be described in terms of the structure-behavior-function (SBF) models that are used in KRITIK to model physical devices. Just like work on KRITIK led to the development of a SBF language for representing and organizing knowledge of the functioning of a physical device, similarly work on AUTOGONOSTIC has led to the development of a related SBF language for specifying knowledge of the reasoning tasks and methods employed by a problem solver in solving a problem. In CANAH-CHAB, we are using this SBF language to specify the design strategies that KRITIK uses. In this way, the design of CANAH-CHAB takes advantage of the task-structure analysis of KRITIK.

Of course merely representing design tasks and problem-solving methods is not enough. The system must also provide the user access to the tasks and methods as and when they are needed. In particular, the system needs to (i) help the user in decomposing a de-

sign task into appropriate subtasks, and (ii) provide access to the design methods that are applicable to a given task or subtask. We use the framework of task structure analysis [Chandrasekaran 1990] for this. In particular, we incorporate in CANAH-CHAB a functional memory of problem-solving methods such as case-based reasoning and model-based reasoning [Goel and Callantine 1991] [Goel *et al* 1993]. A problem-solving method in this method memory is indexed by the tasks for which it is useful. Each method sets up subtasks, and the subtasks in turn act as probes into the method memory.

By having its knowledge of problem-solving methods organized around the design tasks and subtasks they help to solve, CANAH-CHAB can provide access to the problem-solving methods relevant to the design problem specified by a student. Further, since each problem-solving method sets up its own subtasks, it can help the student decompose the given problem. As the student sets up subtasks, the system can help the student in solving each of the subtasks in a similar manner, *e.g.*, the task of design adaptation as illustrated below.

4.1.1 Sample Interaction with CANAH-CHAB

To illustrate how CANAH-CHAB may provide access to generic problem-solving methods, use KRITIK to illustrate them, and guide the user in decomposing a given design task into subtasks, consider the hypothetical scenario of a student's interaction with the system shown in Figure 4.

In this hypothetical scenario, CANAH-CHAB enables the user to understand and learn the generic method of case-based reasoning. It uses KRITIK to illustrate the case-based method with a specific example, and helps the student in using the case-based method for her own problem. For example, it helps the student in setting the subtasks of case retrieval, adaptation, evaluation and storage in the context of her design problem.

4.2 Learning of Design Skills

The design of CANAH-CHAB goes beyond most current tutoring systems and learning environments because it emphasizes the learning of design strategies in addition to domain knowledge as described in the previous section. However, this still leaves open the issue of how to enable a student to become more skilled at using the design strategies and domain knowledge. By this we mean, how to enable the student to recognize interdependencies among design decisions, to recognize errors of reasoning, to recognize faulty design products, etc. These abilities are needed so that the student can incrementally become a more efficient designer, one who can anticipate and avoid errors of reasoning, and produce better quality designs. Note that there is no easy way of incorporating this knowledge directly into CANAH-CHAB because this knowledge is incomplete, tacit and individualized.

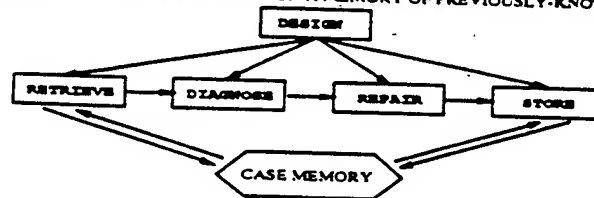
Instead, what is needed is a mechanism for enabling the student to *reflect* on her design experiences. The process of reflection would help the student in transferring CANAH-CHAB's strategic and domain knowledge from the system's frame of reference to her own frame of reference and contextualize this knowledge in her design experiences. In addition, the reflection process would help the student in generating explanations and justifications for

USER:
PROBLEM SPECIFICATION
 I NEED A DEVICE THAT WILL COOL NITRIC ACID FROM TEMPERATURE T1 TO TEMPERATURE T2.
 I HAVE: HNO_3 AT LOCATION R1 AND TEMPERATURE T1.
 I WANT: HNO_3 AT LOCATION R1 AND TEMPERATURE $T1 < T2$.

SYSTEM:
 TO SOLVE THIS PROBLEM, SEVERAL METHODS ARE AVAILABLE: M1, M2, ..., M_n.
 I SUGGEST YOU TRY THE METHOD M2 (CASE-BASED REASONING).

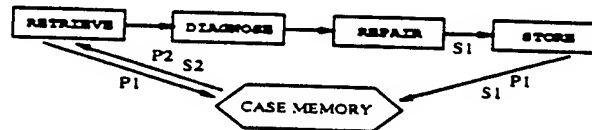
USER:
 EXPLAIN TO ME HOW TO USE CASE-BASED REASONING.

SYSTEM:
 THE SUBTASKS INVOLVED IN CASE-BASED REASONING ARE:
 RETRIEVAL, DIAGNOSIS, REPAIR, STORAGE.
 RETRIEVAL AND STORAGE RELY ON A MEMORY OF PREVIOUSLY-KNOWN DEVICE DESIGNS.



USER:
 ILLUSTRATE HOW TO APPLY CASE-BASED REASONING TO A DESIGN PROBLEM.

SYSTEM:
 CONSIDER THE FOLLOWING PROBLEM P1:
 WE HAVE: ELECTRICAL BATTERY 1.5 VOLTS
 WE WANT: LIGHT - COLOR BLUE
 INTENSITY II
 GIVEN P1'S SPECIFICATION, A CASE COMPOSED OF A PROBLEM P2 AND ITS SOLUTION S2
 IS RETRIEVED FROM MEMORY. DIFFERENCES ARE DIAGNOSED AND THE PROBLEM IS
 REPAIRED TO OBTAIN THE SOLUTION S1. THE RESULT IS STORED IN THE CASE MEMORY.



USER:
 I DON'T REALLY UNDERSTAND WHAT THE DIAGNOSIS OF A PREVIOUS DESIGN INVOLVES.
 EXPLAIN DIAGNOSIS MORE IN DETAILS.

SYSTEM:
 TO DIAGNOSE THE DIFFERENCES BETWEEN THE PROBLEM AND THE RETRIEVED DESIGN,
 SEVERAL STRATEGIES ARE AVAILABLE: S1, S2, ..., S_m.
 THE STRATEGY USED IN THE PREVIOUS EXAMPLE WAS S2 (MODEL-BASED REASONING).

Figure 4: User's interaction with CANAH-CHAB

the design decisions, and strategies for recovering from reasoning errors and design failures. This would help the student in making explicit knowledge of design decisions, reasoning errors, and design failures, which otherwise may remain tacit and unstated.

The centrality of reflection to learning in problem-solving situations has been established by a long line of psychological inquiry, *e.g.*, [Piaget 1971], [Flavell 1971], [Kluwe 1982], [Baker and Brown 1984] [Collins and Brown 1988]. One of the main results of these studies is that reflection upon instances of failed problem solving enables the problem solver to reformulate the course of her own "thinking" so that she does not fail in a similar way under similar circumstances in the future. Similarly, reflection on instances of successful problem solving help the designer to try to repeat this success in similar situations in the future. In the context of design, Schoen [1987] has argued that reflection is a fundamental constituent of design problem solving and learning. Reflection can help a student become a more skilled

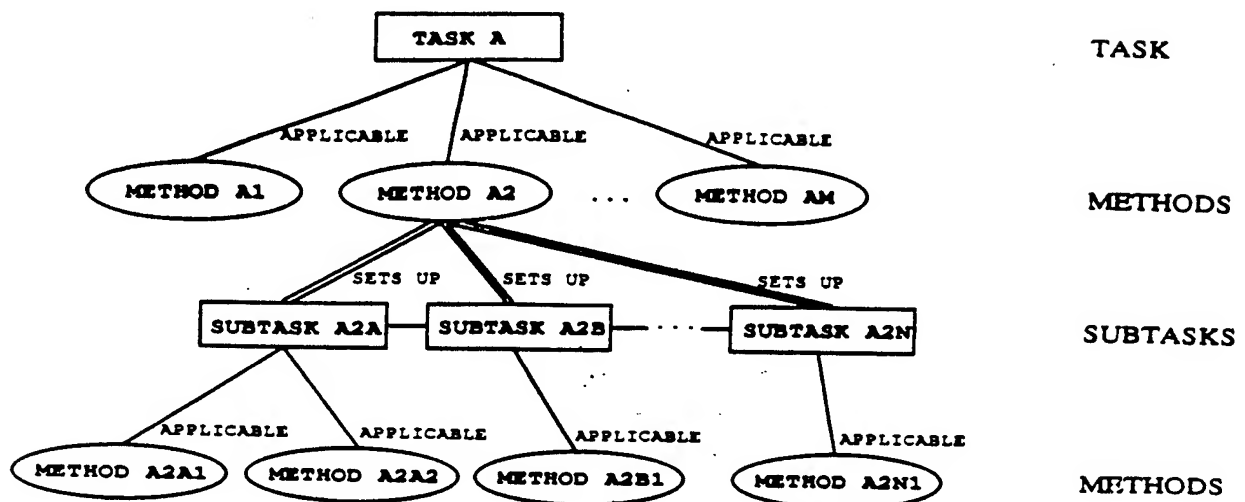


Figure 7: Task-centered component of design rationale (abstract).

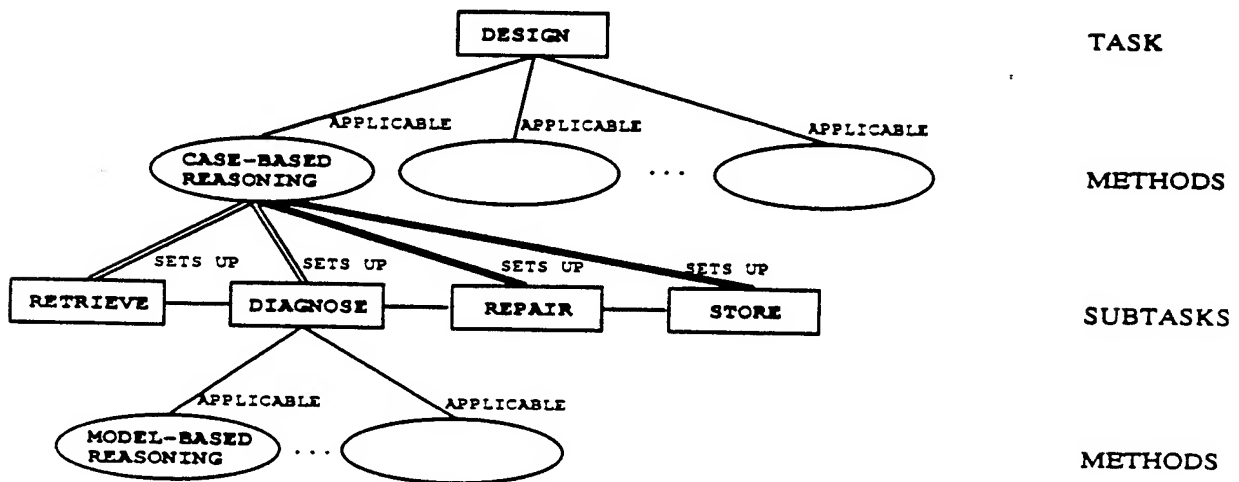


Figure 8: Task-centered component of design rationale (example).

the artifact-centered component potentially enables reflection on the processes of design criticism, redesign etc.

The fourth, issue-centered view, is at "grain-size" smaller than the first three. While it specifies the design rationale at a finer level, it makes no distinction between the three senses of "design," and provides no formal characterization of an "issue." The current design of CANAH-CHAB is intended to support the first three views of design rationale.

5 Current Status

We are currently developing the CANAH-CHAB system along the lines described above. We are constructing CANAH-CHAB on a Sun Sparcstation using the Common Lisp Object System and the Garnet interface development tool. Further, we are building CANAH-CHAB on "top"

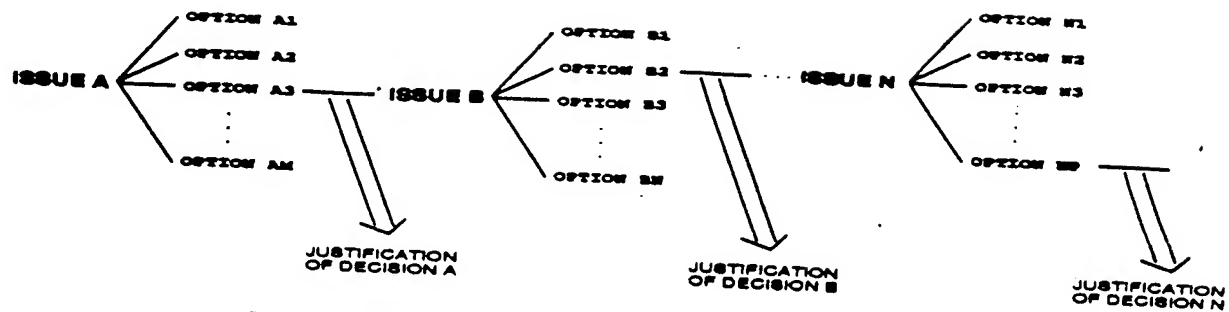


Figure 9: Issue-centered component of design rationale.

of the already operational KRITIK system. This is so that KRITIK can provide CANAH-CHAB with the capability of autonomous design. Intelligent tutoring systems and learning environments typically are incapable of autonomous problem solving. This is because they typically support only the learning of domain knowledge and act primarily as an external memory of this knowledge. In contrast, CANAH-CHAB is intended to support the learning of design strategies and skills in addition to the learning of domain knowledge and design cases. Further, it is intended to use methods such as learning by observing and learning by using for some of these tasks. Hence it needs to have the ability of creating and critiquing designs on its own; KRITIK provides it with this capability.

6 Concluding Discussion

Interactive multimedia computing systems offer a much more flexible environment for the teaching and learning of design skills over traditional "pencil and paper." Flexibility inherent in computing environments offers opportunities for ease of access, networking, and evolution and growth. In addition, this flexibility enables learning in both self-paced individualized and collaborative work environments. A comprehensive interactive multimedia environment potentially becomes the "design studio" that augments, and in some cases replaces, the traditional classroom settings.

These observations have recently led to the development of several experimental computer-based systems for supporting conceptual design (*e.g.*, JANUS, BOGART, and our own earlier work, *e.g.*, ARCHIE, ASKJEF). From the perspective of teaching and learning of design skills, these systems seem limited in at least two ways. First, they provide access to domain knowledge, but not to functional knowledge of generic design tasks and strategic knowledge of generic design methods. Second, they view design primarily as a problem-solving activity, but not also as a learning activity.

Our experience with ASKJEF and ARCHIE TUTOR suggests that before we develop computing environments for enabling the learning of design skills, we need to closely examine the issues of the content and methods of learning these skills. In this paper, we have analyzed some of these issues and have described how our analysis has influenced the design of CANAH-CHAB. In particular our analysis leads us to the following conclusions: